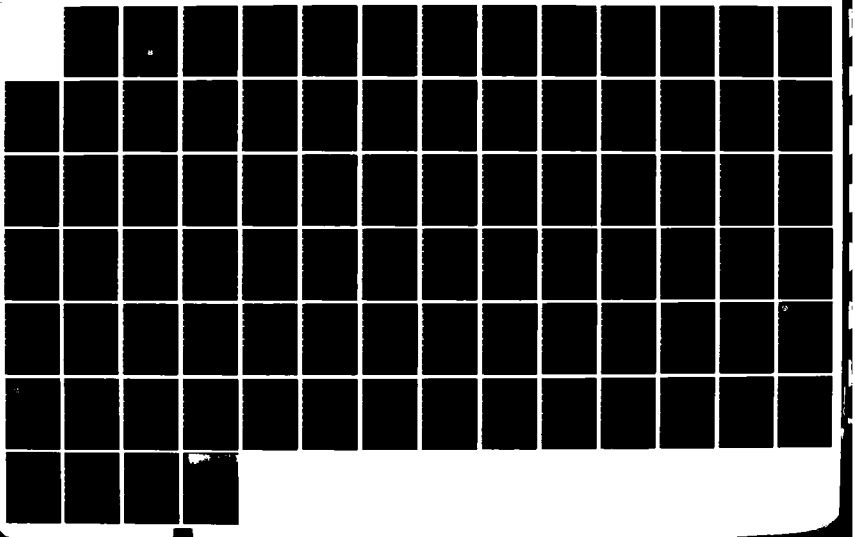
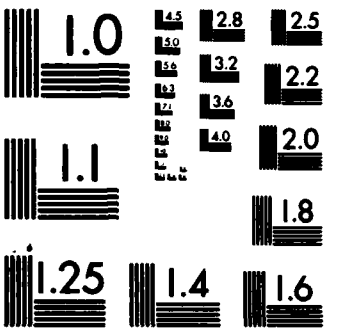
AD-A140 347 PRELIMINARY PROGRAM MANAGER'S GUIDE TO ADA(U) MITRE CORP BEDFORD MA R G HOWE ET AL. FEB 84 WP-25012 ESD-TR-83-255 F19628-82-C-0001 1/1

UNCLASSIFIED F/G 5/1 NL





ESD-TR-83-255 WP-25012

PRELIMINARY PROGRAM MANAGER'S GUIDE TO Ada*

By
R. G. HOWE
W. E. BYRNE
E. C. GRUND
R. F. HILLIARD II
R. G. MUNCK

FEBRUARY 1984

Prepared for
DEPUTY FOR ACQUISITION LOGISTICS AND TECHNICAL OPERATIONS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
Hanscom Air Force Base, Massachusetts

Project No. 5720
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract No. F19628-82-C-0001

* Ada is a registered trademark of the United States Department of Defense

When U.S. Government drawings, specifications, or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Do not return this copy. Retain or destroy.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved for publication.

WILLIAM J. LETENDRE
Program Manager, Computer Resource
Management Technology (PE 64740F)

PERCY L. SAUNDERS, CAPT, USAF
Project Officer, Software Engineering
Tools and Methods

FOR THE COMMANDER

JOSEPH J. FARINELLO
Acting Director, Engineering and Test
Deputy for Acquisition Logistics
and Technical Operations

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) WP-25012 ESD-TR-83-255	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION The MITRE Corporation 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code) Burlington Road Bedford, MA 01730	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Deputy for Acquisition (cont.) 8b. OFFICE SYMBOL (If applicable) ALEE	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F19628-82-C-0001
8c. ADDRESS (City, State and ZIP Code) Electronic Systems Division, AFSC Hanscom AFB, MA 01731	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. 5720 / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification) Preliminary Program Manager's Guide to Ada	
12. PERSONAL AUTHOR(S) R. G. Howe, W. E. Byrne, E. C. Grund, R. F. Hilliard II, R. G. Munck	
13a. TYPE OF REPORT Final Report 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) 1984 February 15. PAGE COUNT 86
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES		
FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

Ada
Government Regulations
Program Management
Risks and Benefits

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This draft guide provides current information that should help a Program Manager in making decisions relative to the use of Ada. It discusses pertinent Air Force and DoD policy, effects of Ada on contractual documentation, and steps that must be taken to apply Ada. This guide identifies benefits and inherent risks of using Ada, and Program Office initiatives needed to control risk. It cites factors that will affect programmer training, software cost and schedule estimation, design, and configuration management. As background information, the basic features of Ada and the software needed to support programming in Ada are described. The guide will be issued after review.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☒ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Susan R. Gilbert	22b. TELEPHONE NUMBER (Include Area Code) (617) 271-8088 22c. OFFICE SYMBOL

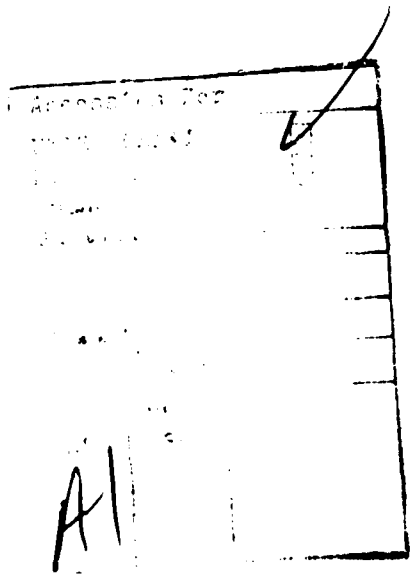
DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

8a. (Concluded)

Logistics and Technical Operations



EXECUTIVE SUMMARY

- Ada is a modern high order programming language developed by DoD for use in large, real-time embedded computer systems. Use of Ada as the standard programming language in mission critical systems is expected to improve the quality of software and reduce the costs for software development and maintenance.

- ANSI/MIL-STD-1815A, which defines the language, was approved in February 1983. A comprehensive set of tests for conformance to the standard has been developed by the Ada Joint Program Office. Only compilers validated by these tests can be used to develop software for DoD.

- DoD has defined requirements for an Ada programming support environment (APSE), a set of software tools used in the development and maintenance of Ada software. Detailed standards and validation tests for APSEs are being developed.

- Ada compilers and APSEs are being developed by DoD and industry; three compilers have been validated and one commercial environment exists. The Air Force's Ada Integrated Environment and the Army's Ada Language System, the DoD funded APSEs, are not yet available.

- DoD Directive 5000.31 will mandate the use of Ada in all mission-critical systems that are new starts in mid-1984 or later.

- The Air Force Systems Command (AFSC) has drafted a transition plan for introducing Ada in four phases, progressing in an orderly manner toward the mandated use of Ada in nearly all systems. This plan indicates that AFSC intends to comply with 5000.31 regulations and will require the use of Ada in AFSARC and DSARC programs. However, AFSC is planning to grant waivers when necessary for meeting cost or schedule commitments or for avoiding excessive risks. The plan calls for the systematic reduction of risk by trial applications of Ada which lead to a mature Ada programming support technology. In the meantime, when Ada is required for use in system developments during the next several years, the AFSC plan suggests that the risks be reduced by either of two means: 1) Advanced Development programs to accelerate the maturity of Ada support tools; or 2) parallel

development efforts involving both Ada and another approved high order programming language. These extra efforts to control risk will be considered to be directed effort and within the scope of the Program Management Directive.

- o Some risks of using Ada in acquisition of operational system software in the near term are in these areas:
  - Limited availability of validated compilers and of APSEs;
  - Lack of information about the performance of the compilers and tools and of the code generated by the compilers;
  - Lack of industry and government personnel with Ada training and experience; and
  - Lack of information with which to predict costs and schedules.
- o Only preliminary guidance is offered now for program managers who must consider Ada. It covers RFP preparation, use of Ada as a program design language, APSE selection and modification, cost estimation and scheduling, configuration management, and training. This guide will be expanded and updated to reflect changes in policy, changes in the status of Ada technology, and the results of Ada experience at ESD and elsewhere.
- o Further preparation for the use of Ada at ESD is necessary. An ESD Ada Introduction Plan will identify required action.

ACKNOWLEDGEMENTS

This report has been prepared by The MITRE Corporation under Project 5720, Embedded Computer System Engineering and Applications, Contract F19628-82-C-0001. The contract is sponsored by the Electronic Systems Division, Air Force Systems Command, Hanscom Air Force Base, Massachusetts.

The authors wish to express their appreciation to the following individuals who gave freely of their time and ideas while reviewing early drafts of this document:

Mr. R. Bowers, GTE-Sylvania, Inc.
Ms. C. Braun, SofTech, Inc.
Major D. Burton, ESD/ALL
Mrs. M. Call, The MITRE Corporation
Mrs. J. Clapp, The MITRE Corporation
Ms. C. Hayden, GTE-Sylvania, Inc.
Ms. M. Hazle, The MITRE Corporation
Mr. G. Kacek, The Raytheon Company
Mr. W. Letendre, ESD/ALEE
Ms. S. Maciorowski, The MITRE Corporation
Captain M. McCormack, ESD/ALEE
Mr. J. Mitchell, The MITRE Corporation
Captain P. Saunders, ESD/ALEE
Mr. O. Shapiro, The MITRE Corporation
Mr. L. Stroup, GTE-Sylvania, Inc.
Mr. J. Van Dolman, GTE-Sylvania, Inc.
Mr. E. Weiss, GTE-Sylvania, Inc.
Mr. H. Willman, The Raytheon Company
Mr. D. Winters, Sanders Associates, Inc.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Concluded)

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION 1

INTRODUCTION

1.1 PURPOSE

Ada is a modern high order programming language intended for use in large real-time computer systems. This report provides information to ESD Program Managers who will be concerned with making Ada-related decisions during FY'84. Only preliminary guidance can be offered now, because of the changing state of Ada technology and the lack of experience in the development of Ada software. This guide will be expanded and updated to reflect changes in policy, changes in Ada technolgoy, and the results of Ada experience at ESD and elsewhere.

1.2 SCOPE

This guide was written with the assumption that the reader is unfamiliar with Ada. Section 2 covers:

Salient characteristics of Ada as a "programming language" (sometimes referred to herein as an "implementation language"),

Support software tools needed to develop programs using the Ada language,

The current development status of Ada programming support tools,

The use of an Ada-based Program Design Language (PDL).

Section 3 discusses the status of DoD regulations relative to the use of Ada. Section 4 describes the Air Force Systems Command plan for introducing Ada. Finally, Section 5 addresses the following subjects:

Factors to be considered while planning for use of Ada in a system acquisition,

Selection of an Ada-based Program Design Language,

Retargeting, rehosting, and validation of Ada programming support tools,

Acquisition and maintenance of Ada programming support tools,

Cost estimation and scheduling,

Training,

Project and configuration management,

Recent experience in applying Ada as an implementation language.

SECTION 2

BACKGROUND INFORMATION

This section describes characteristics and features of the Ada programming language, the support software tools needed to develop programs in Ada, the status and maturity of these Ada-related software development tools, and the use of Ada-based Program Design Languages (PDLs). Section 2 is intended to provide a foundation for understanding the complex set of management decisions, DoD regulations, and planning factors that ESD program managers should consider while acquiring Ada software.

2.1 SALIENT FEATURES OF ADA

2.1.1 The Ada Language

Ada is a general-purpose, high order programming language with considerable expressive power. It was developed under the auspices of DoD for specific use in large real-time embedded computer systems such as C^3I, communications, avionics, and weapon-control systems. These applications require real-time control of many concurrent processes, automatic error recovery and fail-safe execution, and interfaces to many complex nonstandard input/output devices. Ada supports the use of modern software development principles, including top-down structured design, modularity, data abstraction, and information hiding. Ada offers promise for improvements in software maintainability, reusability and portability. It is expected that Ada will soon become formally approved for use in defense projects, and that the use of Ada will become mandatory in 1984 for new DoD acquisition programs involving mission-critical systems.

2.1.2 MIL-STD 1815A

Unlike any other high order language (HOL), the formal definition of the Ada language became standardized (ANSI/MIL-STD 1815A) before compilers were available.[1] In the past, various computer vendors developed compilers that were based on significantly different versions of the same high order language, and these differences tended to frustrate any attempts at standardization. To prevent subsetting or supersetting of the Ada language, the Ada

Joint Program Office* (AJPO) within DoD has registered the name, Ada, as a trademark, and ruled that it can only be applied to "validated" compilers.

In order to become "validated," Ada compilers must successfully pass a comprehensive set of tests, administered by AJPO, which demonstrate strict compliance with the complete Ada language standard. DoD policy requires Ada compilers to be fully validated prior to use in acquisition programs.

2.1.3 Portability and Reusability of Ada Source Code

Use of Ada will promote the movement of software from one type of computer to another (provided that each computer has an Ada compiler), both because the language has been standardized and because implementation dependencies can be localized or isolated. This facilitates both the sharing of Ada software among systems using different types of computers (reusability) and the replacement of computers within a system without major software modification (portability).

Experience with other languages has shown that programmers must learn how to write portable or reusable programs; portability is not ensured by language standardization per se. In addition, the portability and reusability of software will sometimes be sensitive to differences in the characteristics of computers in which Ada object programs will be executing; for example, computational accuracies will potentially be affected when computers have different memory word sizes. Timing problems can arise when there are differences in the efficiency of object programs generated by Ada compilers for different computers.

2.1.4 Software Engineering with Ada

The Ada language contains features specifically intended for functional decomposition and data abstraction, techniques that are increasingly used during the development of large software systems in order to reduce the amount of detail that must be comprehended to understand them. In decomposition, complex software functions are viewed as being composed of collections of simpler functions; each step in decomposition requires that system components be described in further detail (stepwise refinement).

*The Ada Joint Program is responsible for coordinating activities required to produce Ada programming support tools and to promote introduction of the language into participating services and agencies (DoD components).

Proper decomposition enables one to solve problems independently and insures that solving the subproblems also solves the original problem. Abstraction involves conceptual simplification of a complex system by deliberately ignoring detail. In the design of large embedded software systems, software architects have found that decomposition and abstraction are useful concepts for specifying a design that is to be implemented in cooperation with a team of programmers. The complete software problem to be addressed can be decomposed into subproblems that can then be solved separately in terms of program modules. The requirements for each module are specified at an abstract level sufficient to enable its implementation in accordance with the needs of the overall system. Perhaps the most important features of the Ada language, which will be unfamiliar to programmers accustomed to the design of programs in earlier languages, are those that directly enforce these concepts. The Ada language concentrates one's attention on the design and planning of software interfaces. As a result, programmers should spend more time on the early phase of software design and less on software integration and testing than would typically be experienced while implementing a design in some other high order language.

With Ada, program modules are organized into various types of entities referred to as packages, tasks, and subprograms. This form of organization improves clarity and allows tight control over the visibility of names. Ada is modular in both a logical and physical sense. The logical modularity associates specific operations with each data type and tightly controls the visibility of variables; this means that interface specifications for parameter passing and task entry points can be visible to other programmers, even when the internal data and other implementation details (coding) are not visible. This logical separation ensures that changes to the implementation part of a module do not affect other routines that use a module as long as the module interface specification remains constant. Logical separation is enforced by a physical separation of the code for interface specifications from the code for implementation of the functions and procedures within a module. This enables modules to be recompiled without recompiling the modules that use them. The interface specification section of each Ada program module constitutes the "contract" each programmer has with [illegible] on a large project.

[illegible]anagement point of view, the modern software engin[illegible]nciples embodied in Ada provide increased control over the [illegible]gn process. Using the techniques of decomposition, abstraction, and information hiding, large systems can be effectively organized into successive levels of abstraction. Each level of abstraction is a self-contained set of object types along with the operations defined to manipulate those types. Once a

system has been defined in such terms, the job of implementing each level becomes much more well-defined and hence more manageable. Each level may be verified before proceeding to the next level of decomposition, thereby avoiding the typical situations wherein code at lower levels must be rewritten because of flaws at a higher level in the design.

2.1.5 Maintainability and Reliability

Software written in Ada should be simpler to maintain than that written in either FORTRAN and JOVIAL because of its readability and understandability. The Ada language requires a disciplined programming style with uniformity in format among program modules. Even though an embedded computer system might comprise hundreds of program modules, due to the nesting of modules (i.e., packages, subprograms, and tasks) within modules, it is not necessary to comprehend all levels of detail in order to acquire a basic understanding of a software architecture that is implemented in the Ada language. If properly applied, Ada source code should yield an understanding of a software system design. Owing to its readability, Ada should lessen the problems of finding software errors and correcting them. The Ada language is expected to improve software reliability because of the extensive checking that occurs both at compile time and run time.

2.1.6 Tasking

Ada provides unique features for handling real time concurrent processing, and developers will need to devise methods for properly using these capabilities. In the past, programmers had to exercise a fair degree of direct control over what happened and when in a computer. Ada provides an abstraction of real-time concurrent processing. Ada programmers define "tasks," but don't concern themselves with how tasks can be interleaved by the processors, except when synchronization between tasks is required. If two or more tasks require synchronization, the programmer must specify the points of synchronization via appropriate control statements. The circumstances wherein Ada tasks become synchronized during software execution are referred to as "rendezvous." Initially, at least, many programmers will find the tasking concepts of Ada unfamiliar and will prefer to use the traditional methods for interfacing applications software with an operating system.

2.2 ADA PROGRAMMING SUPPORT ENVIRONMENTS

2.2.1 Basic Concepts

DoD has defined generic requirements* (Stoneman[2]) for an Ada Programming Support Environment (APSE). The APSE requirements are primarily applicable to a host computer on which programs are typically compiled and debugged. The DoD requirements do not specify Ada support tools for a target machine in which applications programs are operationally executed.

2.2.1.1 Ada Programming Support Environment (APSE). An APSE is an integrated set of software tools that support the development and maintenance of Ada applications software on a host computer. DoD requirements refer to the individual support programs as "tools." The Stoneman requirements do not constrain an APSE to be a fixed set of tools. The tool set in an APSE can be modified or extended at any time.

Although an APSE tool set can always be expanded, DoD does require an APSE to contain a Minimal Ada Program Support Environment (MAPSE) that is supported by a Kernel Ada Program Support Environment (KAPSE). MAPSE requirements state the minimal set of tools that are both necessary and sufficient for the development and continuing support of Ada computer programs. The KAPSE provides data base, communication, and other services (using the host computer's operating system) that are necessary to execute programs in the host computer.

DoD requirements specify APSE tool capabilities over and above those encompassed by the MAPSE and KAPSE. They include tools for documentation, project control, system requirements specification handling, and program verification. Certain other capabilities, not mentioned in Stoneman, can be developed in Ada and incorporated into the APSE (e.g., an environmental simulator). They have to be requested and evaluated on an individual system acquisition basis.

DoD requirements for the APSE apply almost exclusively to the host computer and not to the target machine. Capabilities for debugging, performance monitoring, and data collection in the target machine are not included in Stoneman and will not necessarily be available in an existing APSE. The program manager may be obliged to expend additional funds to obtain such tools from a software vendor.

*Perhaps the term, "requirements," should not be applied when references are made to the Stoneman document because there's no intention to enforce compliance with the contents of Stoneman.

2.2.1.2 MAPSE. A MAPSE consists of tools that can be considered both minimal and comprehensive. Minimal implies that no smaller set of tools is adequate for Ada software development. Comprehensive implies that it is possible for all members of a project team to work entirely within the MAPSE at all stages of the Ada software life cycle. DoD's reason for defining the MAPSE was to minimize programmer retraining by virtue of having a minimal standard set of tool capabilities available from one project to the next using different computers.

The MAPSE tool set specified by DoD requirements is the following:

- Text editor (for entering and modifying Ada code).
- Pretty printer (used to print text in legible formats).
- Compiler (translates Ada source code into object code for target computer).
- Linkers (resolve interfaces between separately compiled modules, forming executable programs).
- Loaders (used to load executable programs).
- Symbolic debugger (snapshot, trace, etc.) for each target computer.
- Static analyzers (set-use, cross-reference maps, calling relationships).
- Dynamic analysis tools (frequency analyzer, timing analyzer).
- Terminal interface routines (interface terminals with KAPSE software).
- File administration utilities (provides file comparison, file transmission, etc.).
- Command interpreter (accepts commands to invoke tools).
- Configuration manager (to provide rudimentary facilities for software configuration control).
- Stub generator (to insert dummy software elements, in place of expected functional elements, until the expected elements become available).

2.2.1.3 KAPSE. A KAPSE provides data base, communication, and other services necessary to execute Ada computer programs (in the host computer), including MAPSE tools. The data base capability can be viewed either as a sophisticated file system or a rudimentary data base management system. KAPSE software supplements the host operating system to provide a complete set of services that the host operating system would ordinarily provide. In essence, the KAPSE provides a "virtual operating system" interface to the remainder of the APSE tools.

The KAPSE is that part of the APSE that must be re-implemented when rehosting the APSE. Portability* of tools is promoted by ensuring that KAPSE services are identical in form and in substance on all hosts. Moreover, different operating systems running on the same computer (e.g., VMS** and UNIX*** on the VAX computer) will need to be encapsulated by different KAPSEs.

The Stoneman document establishes requirements for configuration management that are so pervasive that support for configuration management has been included in the KAPSE. For example, the KAPSE data base management provides a mechanism to implement a data file manager that stores project-unique data. This device, among other things, permits control over the consistency and correctness of different versions of Ada applications software. General APSE configuration management capabilities are described in Section 5.6.

2.2.1.4 Programming Support for a Target Computer. Computer program integration testing, including interaction with hardware and perhaps an environment simulator, is accomplished on a target machine, where debugging and performance monitoring capabilities should be available. When the target machine and host machine are the same, all of the APSE tools will be available to support both individual-module testing and integration testing. When the target machine is different from the host, the compiler must contain code generation facilities for the target machine. There must also be a loader and possibly a linker, an assembler, a debugger, and timing/frequency analyzers for the target machine.

---

*The term "portability" refers to the ease with which the operation of software can be moved either from one type of computer to another type of computer, or else to the same type of computer under the control of a different type of operating system.

**VMS is a trademark of Digital Equipment Corporation.

***UNIX is a trademark of Bell Laboratories.

DoD requirements do not rigorously specify capabilities for testing software in the target machine. Therefore, capabilities for software testing in the target computer may not be included in APSEs.

2.2.2 Ada Programming Support Environments Under Development

The U.S. Army and the U.S. Air Force have sponsored the development of Ada Programming Support Environments, patterned after Stoneman requirements, with Ada compilers hosted and targeted to the computers shown in table 1. Various commercially-funded Ada tool development efforts are also underway. In the next several years, there will be relatively few Ada programming support environments targeted for the types of computers typically employed in C^3I applications. None have been tested in the acquisition of a military system.

2.2.2.1 The Ada Language System (ALS) of the U.S. Army. The Ada Language System being developed under Army sponsorship by SofTech, Inc., includes a MAPSE, and a KAPSE that interfaces with the VMS operating system in the Digital Equipment VAX 11/780 computer. Initial work has started to retarget the ALS Ada compiler to the Army Military Computer Family computer based on the standard 32-bit instruction set architecture specified in MIL-STD 1862, which is also known as "Nebula."[3]

2.2.2.2 Retargeting of the Ada Language System (ALS). The Air Force has sponsored the retargeting of ALS for the Intel 8086 (in support of the MEECN Program). Requests for proposals have been issued for retargeting of ALS for at least one member of the Navy family of computers, (AN/UYK-20(44)).

2.2.2.3 The Ada Integrated Environment (AIE) of the U.S. Air Force. The Ada Programming Support Environment being developed under Air Force sponsorship by Intermetrics, Inc., with program management by the Rome Air Development Center (RADC), is called AIE. It includes a MAPSE, and a KAPSE that interfaces with the UTS operating system (an Amdahl UNIX-like operating system running in a virtual machine under VM/370 on an IBM 4341 computer).

Table 1

Ada Compilers Under Development by the U.S. Army
and U.S. Air Force

Responsible Agency	Host Computer and Operating System	Target Computer and Operating System
Army (ALS)		
CECOM	Digital Equipment VAX 11/780 and VMS	Digital Equipment VAX 11/780 and VMS
CECOM	Same	Military Computer Family
Air Force (AIE)		
RADC	IBM 4341 and IBM VM	IBM 4341 and IBM VM
Air Force (Other)		
AFATL	Cyber 176 and NOS/BE	Zilog 8002
ESD	VAX 11/780 and VMS	INTEL 8086

2.2.2.4 Other Ada Tools under Development. The Air Force Armament Test Laboratory (AFATL) is managing the development of an Ada Programming Support Environment, which includes a full Ada compiler (written in PASCAL by Florida State University), hosted on the Cyber 176 computer, using the NOS or NOS/BE operating system and is targeted to the Z8002 computer.

Efforts underway at the ROLM Corporation, Western Digital, Telesoft and Intel are examples of commercially-funded Ada-tool development efforts that are either nearing completion or else have recently been validated. (See table 2.) Other commercially-funded efforts, too numerous to mention, are also in progress.

2.2.3 Development Status of Ada Programming Support Tools

2.2.3.1 ALS and AIE. Figure 1 depicts schedules for the ALS and AIE development efforts. The dates in Figure 1 refer to when compiler validations and other acceptance tests have been passed. These dates should not be interpreted to mean that a compiler has been applied, either on an experimental basis or in an actual system acquisition, nor should they necessarily suggest when a compiler is expected to become "mature." Both the ALS and AIE will have to be modified to comply with DoD standard KAPSE requirements (for tool portability to another environment) if these requirements are approved* as a "standard." The scope of this modification effort is not clear and no schedules have yet been established, but it is anticipated that this work will start early in 1986. Work on the AIE MAPSE is currently suspended in order to focus attention within Intermetrics on the AIE compiler; as a result, early releases of the AIE compiler will probably be accompanied by whatever off-the-shelf support software tools are available, and delivery of the AIE MAPSE software will be delayed.

2.2.3.2 Commercial Ada Tools

The Ada Work Center of the ROLM Corporation (with multiterminal Ada software development capabilities) and the Ada support tools for the personal computer of the Western Digital Corporation have been fully developed and both Ada compilers have been validated by AJPO. Other Ada Programming Support Environments will probably be developed within the next two or three years, with perhaps as many as six compilers becoming validated during FY84. However, many of

*Standard KAPSE requirements are being specified by the Kernel Ada Programming Support Environment Interface Team (KIT) under sponsorship of AJPO.

Table 2

Examples of Commercial Ada Tool Development Efforts

Contractor	Product	Status
ROLM Corporation	A hardware/software combination with multiterminal capability, including an Ada development environment patterned after DoD APSE requirements with an Ada compiler targeted to ROLM computers (e.g., the MSE/800 computer, a militarized version of the Data General Eclipse).	Development complete; Ada compiler validated
Western Digital	A personal computing system, including hardware and a software development environment with an Ada compiler targeted to Western Digital minicomputers.	Development complete; Ada compiler validated
Intel	Ada compilers for both the iAPX-432 system of Intel and the Intel 8086 family of computers.	Under development
Telesoft	Ada compiler targeted for MC68000 of Motorola, available with either UNIX or ROS operating system. Can be hosted on the VAX, IBM 370/VMS, and MC68000 computers.	Compiler is currently available for a subset of the Ada language. Compiler for the full Ada language is expected to be validated during FY84.
Telesoft	Ada compiler hosted and targeted for the VAX computer.	Compiler is currently available for a subset of the Ada language. Compiler for the full Ada language is expected to be validated during FY84.

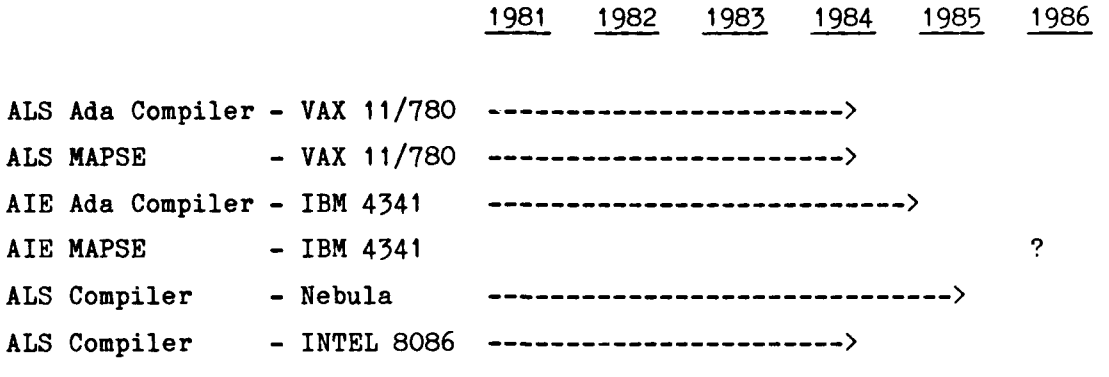


Figure 1. ALS and AIE Development Schedules

the commercially-funded Ada compilers currently only handle subsets of the Ada language standard and therefore cannot be validated. DoD policy precludes the use of these subset compilers in acquisition programs. The commercially-funded Ada tool developments do not really attempt to comply with the Stoneman "requirements;" the commercial tool sets typically contain less than the full MAPSE. Certain compilers targeted for personal computers are reputed to severely limit such things as the maximum number of lines of source code per separately-compiled module and the maximum length of a character string.

2.3 THE USE OF ADA AS A PROGRAM DESIGN LANGUAGE

DoD is encouraging the use of an Ada-based program design language, (PDL), even before the mandate of Ada as the common DoD programming language in mission critical system acquisitions, in the hope that Ada-based designs will be in place for implementation when compilers become available.

2.3.1 PDL Defined

A program design language is a design instrument used to facilitate the conversion of functional specifications into computer instructions. Intended to be comparable to the blueprint in hardware, it can be used to express, communicate, and document the design of a computer program. The term, "PDL", has come to mean a design language for use at the Computer Program Component (or lower) level of detailed design. It should thus be distinguished both from (higher-level) system design languages that are sometimes used for A Level and B Level specifications and from "programming languages" used to implement programs.

A PDL may be used by an individual designer or programmer as a working language for expressing design ideas. When the designer and the programmer is the same person, the development of a PDL specification is a step toward coding of a computer program module. When the designer and the programmer are not the same person, PDL can serve as the language for communication between the designer and the programmer. In the event of changes to requirements or other problems, the PDL specification is the focus of their discussions. In any case, PDL is a language for communicating with software maintainers. PDL may be used as design documentation throughout a project. From a PDL design documentation base, status information may be derived for both technical and management coordination of design and implementation teams. A PDL may also be a suitable medium for formal design review and design verification.

A PDL is used with a set of conventions for depicting certain characteristics of a computer program module. There is a wide range of opinion about both what these characteristics should be and what ought to be included in a detailed design. Ideally a PDL should make it possible to express at least these characteristics:

- Structure and logic of the application (in terms of objects, events, processes, and external interfaces).
- Module structure (defines modules and their interfaces at lower levels than functional architecture).
- Names for data, programs, etc. (to prevent name conflicts and promote understandability).
- Control structure and control flow.
- Data structure and data flow.
- Performance (timing, sizing, resource utilization).

A PDL may be supported by automated tools that check for consistency and completeness in PDL statements.

2.3.2 Ada-Based PDL

Ada-based PDLs are currently planned for use in several ESD programs (e.g., SPADOC, WIS). As indicated in table 3, Ada supports

Table 3

Relationship of Ada Features to Design Characteristics

Level	Design Characteristics	Ada Features
1.	Structure and logic of the application	package and subprogram specifications, private types, and tasks.
2.	Module structure	compilation units and objects.
3.	Naming	identifiers and typing rules.
4.	Control structure and control flow	statements, control constructs, subprograms, tasks, and exceptions.
5.	Data structure and data flow	data types, objects, and exceptions
6.	Performance	none

most of the characteristics needed in a PDL. Since it is text-oriented, Ada can be used for writing selected portions of (C-level) design specifications that can be processed directly by machine for design analysis (consistency, completeness), document preparation, and maintenance.

2.3.2.1 Benefits. Among the benefits of using an Ada-based PDL are the following:

- When an Ada-based PDL is applied selectively, interspersed with English text and other representations in a software design specification, it can be a very effective means for communication among programmers.
- Perhaps the best justification for using Ada as a PDL is that it is a convenient method for expressing interface specifications between program modules. Not only does Ada provide the means to express software interfaces with great precision, it also enables one to automatically test for the completeness and consistency among these interface specifications (via the compiler).
- Features of the Ada language support the definition of major portions of a software design. Using Ada as a basis of a PDL eliminates or reduces the need to document in another language for that purpose.
- When Ada is also the coding language, the PDL evolves into the actual code for the software as details are added.
- If Ada is not the coding language, an Ada-based PDL may still be a good basis both for documenting the design and for reimplementing the code in Ada at a later time when the code is either updated or moved to new hardware or else used in another system.
- Use of an Ada-based PDL will encourage contractor and program management personnel to become familiar with the Ada language, accelerate compiler maturity (assuming that PDL statements will be processed by Ada compilers), and thereby promote the introduction of the language.

2.3.2.2 Risks. There are some risks with using an Ada-based PDL, such as the following:

- Program managers, hardware designers, and communications engineers who are unfamiliar with PDL may find it difficult to review design documentation that consists largely of PDL.

- There is a tendency to include too many details in the design too soon when a PDL includes the capabilities of the full Ada programming language. The contractor's methodology needs to carefully define how a PDL will be used.

- Training costs and costs to maintain PDL statements (in addition to implementation language coding) are likely to increase the overall costs of software development.

- If the coding language is very different from the PDL, then the transition from PDL to code may become a difficult process that is prone to error. (This observation applies to any PDL, not just to Ada-based PDLs.)

- If the implementation language and the PDL are the same (i.e., full Ada), programmers will tend to begin coding before the design is complete and verified, and the design will evolve in the form of increasingly detailed threads of functionality rather than as complete descriptions of the system at each level. Proper management discipline is needed to circumvent this potential problem.

- Although a PDL is useful for describing software-software interfaces, it is not appropriate for defining hardware-software interfaces.

SECTION 3

CRITERIA FOR USING ADA ON AIR FORCE PROGRAMS

Program Offices acquiring software for Air Force systems specify the requirements for the programming language(s) to be used. The choice of language will be influenced by pertinent Air Force policies and regulations, the characteristics of the system, and the candidate language technologies. What follows is a discussion of the factors a program manager should consider in deciding whether to require the use of Ada as an implementation language.

3.1 REGULATIONS AND THE NEW ADA MANDATE

The regulations relating to the use of programming languages in Air Force systems are changing. The current DoD and Air Force regulations seek to minimize the number of different languages used, limiting the selection to a small set of standard high order languages that does not include Ada. However, new regulations are in process that mandate the use of Ada for many defense systems, and the trend in the DoD regulatory climate is towards broadening the scope of the Ada mandate and encouraging its use for all systems. As a result, Program Offices entering an acquisition in the near future may be subject to regulations that either prohibit, encourage, or mandate the use of Ada, depending on the rate of change of the regulations and the type of system being acquired.

The existing DoD Directive 5000.29, the Management of Computer Resources in Major Defense Systems[4], requires that only high order languages approved by the DoD be used to develop defense system software, unless it is demonstrated that none of the approved languages is cost-effective or technically practical for a given system. A separate regulation, DoD Instruction 5000.31[5], specifies the list of six DoD approved languages. Air Force Regulation 300-10[6] and its Air Force Systems Command supplement further limit the languages that can be used in ESD acquisitions managed under series 800 or 300 Air Force regulations to COBOL, FORTRAN, and JOVIAL.

A draft revision to DoD Instruction 5000.31, containing the approved DoD language list, is being circulated for final concurrence and signals the mandated use of Ada; refer to the Appendix. It has been retitled, Computer Programming Language Policy, and elevated from a DoD Instruction to a Directive. It contains a new list of DoD standard languages including all of the standard languages in

the old list except two (SPL/1 and TACPOL), along with Ada. In addition, it requires that Ada become the single common programming language for new "mission critical" applications, effective 1 January 1984 for Advanced Development Programs, that is, programs funded under 63 program elements, and 1 July 1984 for programs entering Full Scale Development. These dates refer to when the requests for proposals (RFPs) are issued rather than the dates when contracts are awarded. "Mission critical" systems in this context include computer resources for research and development of or use in:

a. The National Military Command System (NMCS), the World Wide Military Command and Control System (WWMCCS), DoD Component command systems;

b. Intelligence activities for the intelligence community;

c. Cryptologic activities authorized by the National Security Agency; and

d. Weapon systems -- as integral parts of the systems themselves, or for training, diagnostic testing and maintenance, simulation, calibration, and research and development of weapons systems.

The directive also stipulates that only compilers validated by the Ada Joint Program Office can be used to develop software to be delivered to or maintained by the government. The draft 5000.31 Directive was expected to be formally approved in September, 1983, but was delayed due to pressure from within DoD to expand the scope of the Ada mandate to include Automatic Data Processing (ADP) Systems.

The new 5000.31 is a strong mandate of Ada for mission critical systems and in time will precipitate revisions to the component service regulations in order to require Ada, perhaps even for nonmission critical applications. Historically, ESD has managed many acquisitions for mission critical systems, and therefore it is anticipated that numerous ESD programs will fall under the Ada mandate within the next several years. Currently, Ada is being used as the implementation language in only one ESD acquisition program. Sonicraft, Inc. (Chicago, Illinois) is developing VHF/LF diversity reception Ada software in support of the MEECN Program, Project 616A.

AFSC has prepared an Ada Introduction Plan, described in section 4, that defines the conditions suitable for the use of Ada in AFSC programs.

3.2 FACTORS TO BE CONSIDERED IN DECIDING WHETHER TO USE ADA

In deciding whether to apply Ada, the program manager should exercise care to avoid unacceptable risk to his program. Characteristics of the program and the status of Ada are the two major determinants. As important as it is to begin to apply Ada, it is equally important for the program manager to establish projects that have a reasonable probability for successful completion within schedule and budget. To this end, the program manager should consider the following factors in deciding whether to require the use of Ada versus another approved HOL in an Air Force program:

- o Whether the system being developed falls under the Ada mandate.
- o Whether usable validated compilers exist that generate code for the operational computers selected for a new program. "Usable" means with adequate documentation, meaningful error messages, and producing acceptably efficient and reliable programs.
- o Whether there are usable Programming Support Environments.
- o Whether there are, or will be, facilities and resources for maintaining the candidate compilers and environments.
- o Whether Ada technology and experience has matured sufficiently to moderate the risk and cost of using Ada to the point that is consistent with the scale and priority of the system to be developed.

3.2.1 Good Candidates for Use of Ada Now

To be amenable for software development in Ada now, an acquisition program should have the following characteristics:

- o Can use commercial micro/mini computers for which commercially-developed Ada compilers are or will be available soon.
- o Uses a computer that is ground based and without power, weight, or size constraints; has a large address space; and has a family of processors available so that migration to a larger model is possible.
- o Has a modest processing load and is therefore not expected to require most of the CPU throughput, (i.e., there will be plenty of spare CPU computer capacity).

- Does not have critical software timing requirements.
- Does not have a critical schedule that can't afford any slips.

3.2.2 Bad Candidates for Use of Ada Now

Program managers are advised not to use Ada at this time for systems that are characterized as follows:

- Applications where computer size, weight, power, and speed are fixed and the computer cannot be easily expanded or upgraded.
- Applications where a new compiler or code generator must be prepared, unless there is adequate lead time (estimated at 2 years) for development.

The issue of whether or not an acquisition program should require the use of Ada, if it is not mandated to do so, is a matter of risk versus benefits.

3.3 BENEFITS OF USING ADA AS AN IMPLEMENTATION LANGUAGE

Some of the benefits of Ada are attributed to the commonality of programming language and environment across many Air Force programs; other benefits stem from features of the language itself. Most of these benefits may not be realized for several years. The benefits of using Ada now are as follows:

- The Ada language was designed to support capabilities required for software in embedded computer systems.
- The Ada language was designed to support teams of people who are jointly developing large amounts of complex software.
- Ada compilers must be validated by an extensive set of government furnished tests to determine that all features of the language have been correctly implemented. While this is by no means a guarantee that a compiler will produce completely correct code, validation increases that likelihood, and the validation process does not have to be re-accomplished for each ESD program.
- Use of the Ada language should reduce the costs for software maintenance.

The benefits of using Ada in the future are the following:

- o As Ada programming environments become available, they may increase productivity in software development, increase the ability of software managers to control acquisition, and provide better status information to managers.

- o Ada is a standard language, which means that it has a formal, rigorous definition that has been approved by several standards organizations, and changes to the Ada language will be carefully reviewed and controlled.

- o Greater commonality of programming languages among ESD programs will increase the opportunity for re-using applications software, with increased programmer productivity being the result.

- o Many programmers and software engineers proficient in Ada will become available; language-unique training will no longer be necessary.

- o Industry appears to be ready to adopt Ada. Universities are actively establishing courses in use of Ada language.

3.4 RISKS OF USING ADA NOW AS AN IMPLEMENTATION LANGUAGE

The risks in using Ada will change over time. Currently the risks are in the following areas:

- o The performance of the code generated by existing Ada compilers has not been measured for specific applications such as C^3I systems. The reliability of Ada object programs is unknown. Validation does not necessarily mean that a compiler is mature; bugs still exist in validated compilers.

- o Tools for an Ada environment are not readily available; there is no standard for interfaces between the user and the Ada environment and, therefore, the initial use of Ada may be without tools or else with unique (nonstandard) tools.

- o There are few experienced Ada programmers and systems designers who know how to exploit Ada. Extensive training is necessary before a programmer acquires proficiency.

- o Recent experience has shown that there are some problems with using Ada as a programming language (as described in section 5.8).

3.5 RISK MANAGEMENT

The AFSC Ada Introduction Plan[7] has two approaches to management of the risks associated with Ada introduction. The phased, longer term approach is described in section 4. The approach suggested for programs that use Ada in the near term is summarized here. According to the plan, "Management can lessen the risk of using Ada by taking either of the two approaches described below: accelerated maturity or duplicate development. The extra efforts to control risk will be considered (by USAF) to be directed effort and within the scope of the Program Management Directive, and therefore will be included in the program baseline."

3.5.1 Accelerated Maturity

When three conditions are present,

1) a system development program is directed to use Ada,
2) there are significant inherent risks in the program other than the use of Ada, and
3) the required tool maturity, training, application-specific Ada experience, cost and schedule estimation ability are deemed to be inadequate,

then, "the program manager will be required to schedule and fund activities to correct these deficiencies. These activities can include accelerated tool development, additional design studies, exploratory developments, and program office training courses and the like. These activities should be completed prior to starting the actual software system development effort. Throughout the system development effort, the program manager should be alerted to any additional shortcomings as they surface, and must be prepared to delay the main effort while they are corrected. This approach amounts to trading cost, schedule, and program office manpower for risk control."

3.5.2 Duplicate Development

"As described above, the accelerated maturity approach involves significant lengthening of the program schedules to control risk. Some programs that are directed to use Ada will not have this option. Examples are programs whose completion dates have been directed by the President or the Congress. These programs will be

required to press on with Ada with whatever resources are available. In most cases, they will require correction of resource shortcomings concurrent with application software development. This approach to software system development involves extremely high risk unless there is a back-up. To control risk, programs using this approach will be required to fund and manage a duplicate development of all deliverable software in some mature standard language. Duplicate development means the following:

- o It applies to all deliverable software, not just a subsystem;
- o Both efforts are going all-out to meet schedule and technical performance requirements; and
- o Rather than having a predetermined milestone for termination of an effort, both efforts will continue until there is a clear winner. Specifically, the development in the mature standard language will continue at least until an acceptable Ada programming support environment has been completed, the compiler has been validated, and the program manager has acquired confidence that the development in Ada will succeed.[10]

"The duplicate development approach has been used successfully on previous language introductions. For example, on the B-1A in the mid-1970s, JOVIAL J3B won over assembly language in a little over a year. Duplicate development amounts to more than doubling program office software manpower and software development cost to control risk while meeting program schedule and technical performance requirements."

3.6 WAIVERS

There will be occasions when neither of the risk management approaches described in section 3.5 will be appropriate and a waiver not to use Ada will be required. It is AFSC policy[7] to comply with both the letter and the spirit of the new 5000.31 by requiring Ada on both Defense System Acquisition Review Committee (DSARC) and Air Force System Acquisition Review Committee (AFSARC) programs, and by granting waivers only when they are required for meeting program cost or schedule commitments or for avoiding excessive risk. It is the Air Force position that a lack of required Ada development tools is not, per se, justification for a waiver; this policy is intended to accelerate the development of new APSEs. Programs that are directed to use Ada and that face shortcomings in Ada resources are

expected to adopt either of the risk management approaches previously described in section 3.5.

If, after quantifying the adjustments to cost or schedule required for risk management, both risk management approaches are shown to be infeasible, that fact can be used to justify a waiver. The estimated cost and schedule adjustments should be included in the waiver request. Other possible justifications for a waiver include severe requirements for object code efficiency, a requirement to use a processor that is not suitable for Ada (such as an 8-bit microprocessor), or a requirement to interface with existing software written in another language. Waiver requests must include an alternative requested language that overcomes the reason for not using Ada. In any case, the methodology of the risk management approach provides the framework for developing a solid quantitative, defensible waiver request.

Waivers need not be obtained for use of an application-oriented (including problem-oriented) language or for commercially-available off-the-shelf software if no modification of the software is anticipated over the life cycle. For example, Ada will not replace ATLAS as the HOL used for automatic test equipment.

The new DoDD 5000.31 delegates Ada waiver approval authority to the Services, but requires that approved waivers be sent to the Defense Computer Resources Board for review and possible reversal. In the Air Force, the implementing command will have final waiver approval authority. Implementing commands will send approved waivers through HQ USAF/RDX or HQ USAF/SIT (depending on which has Air Staff responsibility for the system) to SAF/AL for transmittal to the Defense Computer Resources Board.

In the beginning, according to the Director of AJPO, DoD has expressed willingness to be lenient about the granting of waivers relative to the use of Ada, but has also indicated its reluctance toward granting waivers that would exempt a program from having to apply one of the other approved HOLs. ESD programs not falling under the Ada mandate will be required to develop software in one of the other approved languages, and based on the current provisions of AFR 300-10 and its AFSC Supplement, these will be COBOL, FORTRAN, and JOVIAL.

If an ESD program manager has advance knowledge that Ada will be inappropriate for his acquisition program, then he should apply for a waiver as early as possible and indicate in his RFP package that the waiver has been approved. In situations where a program manager determines from proposals submitted by various bidders that

Ada is a poor choice for the implementation language, he should defer the contract award until a waiver has been approved.

SECTION 4

AIR FORCE SYSTEMS COMMAND ADA INTRODUCTION PLAN

The new DoD Directive 5000.31 will require each of the services to prepare a plan for introducing Ada and phasing out other languages, and the Air Force has prepared a draft plan that is now the approved AFSC Ada Introduction Plan. Air Force experience shows that the introduction of a new language entails substantial cost and schedule risk to development programs, and the plan provides a four-phased strategy for introducing Ada rapidly, but in pace with the readiness of Ada to support development with acceptable risk. A key feature of the strategy is a set of criteria for deciding when the technology is ready to move on to the next phase, and the criteria for each phase serve as objectives for the previous phase.

4.1 PHASE 1, LABORATORY DEVELOPMENTS AND EXPLORATIONS

Phase 1 consists of efforts to develop Ada tools and investigate the use of Ada in the application areas in which the Air Force laboratories specialize. The efforts will be done by the laboratories, by industry under contract to the Air Force, or independently of the Air Force.

Since this is the first phase there are no formal criteria for start-up of a program, other than that it address a legitimate need and be initiated and funded through normal laboratory channels. The objectives are to mature compilers and other Ada tools so they can be used for Phase 2 programs, identify practices for Ada use in each application area, and to obtain manpower statistics for sizing Phase 2 efforts.

4.2 PHASE 2, PRODUCT DIVISION PARALLEL OPERATIONAL SYSTEM DEVELOPMENTS

Phase 2 consists of developing Ada code for moderate size applications in parallel with its development in an advanced or full scale development program, in a more mature language. Phase 2 programs will be performed by the product divisions, the laboratories, or the AFLC. The AFSC embedded computer resource focal points will nominate Phase 2 candidate programs.

The criteria for start-up of a Phase 2 program are that the compiler is tested against the Ada validation suite, that it has

previous use that substantiates its maturity, and that it has a minimum set of complementary development tools. In addition, documentation and a maintenance capability must be available for the compiler and tools, along with manpower statistics adequate for sizing the Phase 2 effort.

The objective of Phase 2 is to facilitate the transition of Ada from the laboratories to the product divisions and further mature Ada tools, without putting operational system developments at risk.

4.3 PHASE 3, USE OF ADA ON SELECTED OPERATIONAL SYSTEM DEVELOPMENT PROGRAMS

Phase 3 consists of the first use of Ada for the development of operational systems, in small, low risk programs, and may be carried out by any Air Force organization with a development charter. Programs participate in Phase 3 either by being volunteered by their Program Office, with implementing command approval, or by direction of an implementing command.

The criteria for start-up of Phase 3 programs are that the compiler to be used must be validated, that the tool set meet all of the Phase 2 requirements and include a symbolic debugger and measurement capability, and that the tool set be documented to MIL-STD-483/490 or equivalent. Further, a quick reaction maintenance capability and life cycle support must be in place for the tools, and the selected system contractor must have a method of training the people in designing, coding, and testing Ada programs.

The objective of Phase 3 is to mature Ada tools and Air Force experience towards mandated use of Ada, through application to operational systems that can accommodate the risk.

4.4 PHASE 4, USE OF ADA ON NEARLY ALL PROGRAMS

Phase 4 consists of putting regulations in place, at the appropriate time, to mandate the use of Ada for programs below the AFSARC threshold. In combination with the new DoDD 5000.31, which mandates the use of Ada in AFSARC and DSARC programs, these regulations will make Ada mandatory for all Air Force programs.

The criteria for making Ada mandatory are that there are sufficient proven compiler host/target computer combinations to support most Air Force applications, that they have been successfully used on previous operational system developments, and are supported by an available quick reaction maintenance and other life cycle support

capabilities. Also, there must be more than one source for compilers and tool sets for each application area, and the Air Force and contractors must have enough productivity data on previous programs to estimate future program costs and schedules.

The objective of this phase is to implement an Ada mandate consistent with the Air Force's obligation and commitment to facilitate the timely transition to the standard language, without jeopardizing critical system procurement costs and schedules.

SECTION 5

PLANNING FOR SYSTEM ACQUISITIONS ENTAILING THE USE OF ADA

This section addresses several topics that program managers should consider when planning the use of Ada in an acquisition program. It is organized as follows:

- o Preparation of System Performance Requirements, Request for Proposals (RFPs), and Instructions for Proposal Preparation (IFPPs)
- o Use of PDL
- o APSE Portability, Retargeting, Rehosting, and Validation
- o APSE Selection, Acquisition, and Maintenance
- o Cost Estimation and Scheduling
- o Project and Configuration Management
- o Training
- o Current Problems Using Ada as an Implementation Language

5.1 PREPARATION OF SYSTEM PERFORMANCE REQUIREMENTS, RFPS, AND IFPPS

If a program manager decides to levy a requirement on a development contractor to use Ada, then there are many things to be considered during the preparation of the RFP, the IFPP, and the System Performance Requirements Specifications.

5.1.1 System Performance Requirements Specification

The requirements to use Ada as the implementation language will typically be specified in section 3.3.8 of the System Performance Requirements Specification. In this same paragraph, the program manager may also choose to specify requirements for the use of an Ada-based PDL; however, owing to the ill-defined nature of Ada-based PDLs, it will be rather difficult to specify meaningful requirements for PDLs that can serve as the basis for determining compliance on the part of the contractor. For some system acquisitions, specific guidelines should be specified in Paragraph 3.3.8 for purposes of achieving reusable, portable software. Finally, Paragraph 3 should require that the target machine be selected from a family of computers with upward-compatible growth capabilities in order to reduce the risks associated with inefficiencies in Ada-compiled code.

5.1.2 RFP/SOW Preparation

In the next couple of years, information about programming support tools that program managers might normally expect to have on hand will simply not be available for Ada. For this reason, it is suggested that in the SOW associated with a RFP, the program manager should consider requesting the following work as part of a system acquisition:

- Task to select an appropriate APSE; include performing benchmark tests on selected Ada compilers in order to determine object code efficiency and compile-time performance.
- Task to retarget a compiler and/or Ada programming support tools, if applicable.
- Task to develop customized run-time support libraries to meet system performance requirements, if applicable.
- Task to prepare a timing and sizing study, as a separately deliverable document, that is to be updated several times during the course of the acquisition program.
- Task to prepare a detailed configuration management plan that demonstrates understanding of Ada's unique needs and capabilities (see section 5.6).
- Task to formally demonstrate that the Ada Programming Support Environment tools work correctly; this demonstration will occur at the outset of the acquisition program.
- Task to prepare a Computer Program Development Plan (CPDP) that shall include the following information in addition to other CPDP data:
  - Reserve processing capacity that will be set aside due to the potential for unexpected inefficiencies in Ada-compiled programs.
  - Programming support tools to be procured by the contractor for the host and target computers.
  - Programming support tools to be developed.
  - Proposed design and programming standards and conventions for use with Ada.

- Schedule allowances and plans made by the contractor for training personnel in use of Ada.

- Fallback position if a suitable compiler and APSE cannot be found.

- The PDL selected, the procedures for using PDL, and the manner in which PDL will be included in formal documentation.

o Task to maintain Ada Programming Support tools.

5.1.3 IFPP Preparation and Source Selection Criteria

5.1.3.1 Evaluation of a Bidder's Software Development Organization and Approach. The IFPP should require the bidders to provide information in their proposals that will allow the Air Force to determine their relative Ada software development capabilities. This should include a description of the programming organization (including all subcontractors), capabilities, experience, and an analysis of the proposed approach to minimizing the risks inherent in the use of the new language technology. Descriptions of the bidder's proposed use of a program support library, program design language, and programming organization, as they relate to Ada technology, should be required. Since it is expected that bidders on early Ada developments will have little experience in using Ada, the bidders should be required to describe their plan for training personnel in its use. In addition, the IFPP should require that bidders describe their general approach to software development and configuration management using Ada.

To help evaluators assess the bidder's software development organization and technical approach, a program office may request the bidder to address the following topics in his proposal:

o Contractor's assessment of the suitability and risks (technical, schedule, and management risks) of using Ada.

o Previous Ada developments undertaken by the bidder, including number of lines of code developed and the methodology used.

o The level of training and experience in use of Ada for key software development personnel (e.g., the number of Ada courses taken, previous Ada programming efforts, etc.).

o The amount and type of training in the Ada technology to be provided by the bidders to programmers inexperienced in Ada.

- o The bidder's grasp of the software engineering disciplines supported by Ada, as manifested in his proposed methodology.

- o Contractor's previous performance in acquiring software development facilities: selection, acquisition, and integration.

- o How the contractor intends to provide adequate management and technical control and visibility to the government during the development of Ada software.

- o Rationale provided for selecting a particular Ada-based PDL, if applicable.

- o The manner in which the contractor intends to include Ada-based PDL in C-level design specifications, if applicable. (For example, will the PDL encompass a large amount of the design requirements or will it largely be limited to places where flow charts might otherwise have been provided?)

- o The contractor's plans for acquiring, installing, and maintaining APSE software.

- o The contractor's approach for maintaining configuration control over Ada software development, including the use of automated tools, especially in situations where software development is expected to occur at more than one location.

5.1.3.2 Evaluation of Software Development Facilities. The IFPP should also require that the bidders describe the facilities to be used in developing Ada software, including Ada programming support tools and host/target computer hardware facilities. The proposal should specifically address the maturity of the compiler and programming support environment software proposed. Also the quality of the development facilities should be identified as source selection criteria. Some of the factors a program office should weigh when considering the proposed facilities are:

- o The characteristics of the host/target computer combination, and number of hosts required for effective use of the proposed APSE (e.g., to provide adequate user response time).

- o The reliability of the compiler, the amount of the software that has been developed using it, its performance (e.g., the number of Ada statements it compiles per minute), the efficiency of the object-code generated by the compiler, and

whether it has been validated. The bidder should be required to indicate how this data was obtained.

- o The usefulness of the source-code error diagnostic messages produced by the compiler.
- o The capabilities of the text editor and other programming support tools provided for Ada code development.
- o Life cycle support available for the compiler and all other tools.
- o Documentation of all of the tools, including the compiler, to MIL-STD-483/490 or an equivalent standard.
- o The number of console terminals that can be supported simultaneously while running independent compilations, including how this data was obtained.
- o Debugging capabilities while testing in both the host and the target processors.
- o Availability of tools to support the use of Ada-based PDLs.
- o A quick reaction maintenance capability for the Ada compiler and other software development tools.

5.1.3.3 Cost Proposal Preparation. Separate estimates for designing with PDL and for developing lines of source code in Ada should be provided, and the rationale behind the programmer productivity factors used in preparing these estimates should be explained. If certain portions of the Ada applications software were to be borrowed (reused) from another acquisition program, then the cost estimates for introducing this reusable software should be disassociated from the estimates for developing new lines of source code.

5.2 USE OF PDL

The program manager will need to decide whether the use of PDL will be required. If PDL is to be required, then additional decisions must be made as follows: whether or not the PDL should be based on Ada, whether or not automated aids will be required, and whether or not PDL will be supplemented by other forms of presentation in the formal documentation submitted to the government.

5.2.1 Policy

If MIL-STD-SDS[8] becomes approved, then during the development of both top-level and detailed software design, the contractor shall be required to use a Program Design Language. In the meantime, MIL-STD-483A permits the use of a PDL in place of flow charts in software documentation, subject to the approval of the procurement authority[9].

5.2.2 Selection of an Ada-Based PDL

5.2.2.1 Many Ada-Based PDLs to Choose From. Altogether, there are at least 60 different Ada-based PDLs currently in existence. Efforts are underway in the IEEE Project 990 to provide guidelines for the manner in which Ada is applied as a PDL, but these guidelines are not expected to be available for several years.

Several approaches have been taken to establish Ada-based PDLs. A major consideration is the exact relation of a PDL to the Ada language; some use the full Ada language, some choose an Ada subset, others have defined extensions to Ada. Each of these choices has advantages and disadvantages.

Using a PDL that incorporates the full Ada language provides the designer with Ada's program unit-level features (generics, packages, tasking, etc.) that are useful in design. If Ada is also the implementation language, the transition from design to code is also simplified, since designers and programmers are working in a common language. However, in this case, design and code can become indistinguishable and the use of the full resources of the language should be managed (through a design discipline or methodology) to prevent premature coding from preceding design. If such design methods can be provided, employing full Ada as a PDL has the added advantage of the PDL being processable by Ada compilers. An Ada compiler could serve as a basic design tool for syntax checking, set-used listing capabilities, interface consistency checks, listings, etc..

Some groups are applying various Ada subsets as PDLs. They argue that this reduces the complexity of the language, encourages design to take place at "a higher level" than coding, and makes design easier. In situations where an Ada-based PDL is applied in conjunction with an implementation language other than Ada, there is justification for supporting the notion of using an Ada subset as a PDL because it is difficult to implement certain Ada features (not in the chosen subset, e.g., rendezvous, generics) in other approved HOLs.

Other groups have chosen to extend Ada in a variety of ways -- their rationale being that a richer language will free the designer for design by reducing the burdens of having to write everything in Ada. As an example, some superset Ada-based PDLs include provisions for pseudo-code and for structured commenting, which requires that certain information be incorporated into the design as comments (such as module names, descriptions, interfaces to other modules). These may or may not be processable by an Ada compiler.

5.2.2.2 Criteria for Selecting a PDL. The ESD program manager should consider the following criteria while reviewing a contractor's proposal to use an Ada-based PDL:

- The manner in which a PDL fits into the contractor's overall software methodology. If a PDL will only be used to define software interfaces (Level 1 in table 3, section 2.3.2), then a relatively small subset of the Ada language will be needed for the PDL. If PDL will be used to describe a detailed software design (Levels 1-6 in table 3), then perhaps the complete Ada language should be included in the PDL. Also, the program manager should evaluate the contractor's procedures for avoiding the tendency to produce code instead of design while using a PDL.

- The choice of a PDL can be influenced by the selection of the implementation language (HOL). The programmer has to interpret the design text and produce code that satisfies the intent of the design; he must therefore be fluent in both HOL and PDL conventions. As shown in table 4, certain Ada features can be easily implemented in some HOLs but not in others. Accordingly, one might select a PDL that facilitates design implementation in the chosen HOL. Only a few of Ada's features will easily correspond with those of the implementation language and these tend to be low level features of Ada (such as if then, case) rather than more abstract constructs (such as packages, tasks, and generics). If the implementation language is different from the PDL, the contractor should be asked to document the mapping from PDL to HOL that the programmers are to follow.

Table 4

Relative Implementability of Ada/PDL Constructs in Several High Order Languages

Ada Construct	FORTRAN	JOVIAL	Pascal	COBOL	CMS-2
Tasking/Rendezvous	H	H	H	H	H
Subprograms	M	E	E	E	E
While	E	E	E	E	M
Package	H	M	M	H	H
Generics	H	H	H	H	H
Exceptions	H	M	M	H	E
Data Types	H	M	M	H	M
Case	E	E	E	E	E

Easy: There is a direct correspondence or a simple, regular procedure to guide the implementor.

Moderate: There are potentially several cases to consider and identify; but, for each, some regular implementation procedure may be provided.

Hard: Language differences are such that, in effect, to implement is to redesign.

- The selection of a PDL should take into account the manner in which PDL statements will be included in software documentation that the government will review. Subject to the approval of the procurement authority, MIL-STD-483 has been revised to allow for the use of PDL statements in lieu of "charts" in software specifications. The ESD program manager needs to decide not only whether to include an Ada-based PDL in the C-level documentation, but also how extensively will the PDL statements be included. It is recommended that various examples of C-level documentation as proposed by the contractor be evaluated prior to making this decision. The program manager may find it difficult to review documents that are primarily written in a PDL that is characterized by having a low PDL-to-source code expansion ratio.

5.3 APSE PORTABILITY, RETARGETING, REHOSTING, AND VALIDATION

5.3.1 APSE Portability

The Stoneman requirements are intended to make MAPSE tools portable. However, specifications for the KAPSE-MAPSE interface have not yet been standardized and any tools developed prior to interface standardization will be subject to change. The KAPSE Interface Team (KIT), under direction from AJPO, is currently working on standardization requirements, but the results of these efforts are not likely to go into effect before 1986.

The concept of developing APSE tools in Ada as the implementation language promotes software portability. Although the AIE and ALS APSE development efforts have adopted this approach, many of the commercially-funded efforts have not, and that will adversely affect the portability of those software environments.

Ada compilers translate Ada source code into machine code in several steps. The "front end" of the compiler accepts Ada source code as input, and (after performing syntactic and lexical analyses to determine the correctness of the source code), generates an expanded version of the program in a format commonly referred to as "the intermediate form." This intermediate form is further processed by the "back end" of the compiler to produce object code in machine-language form. Certain APSE tools, most notably the symbolic debugger, operate on the intermediate form of an Ada program. Standardization of the intermediate form would increase tool portability and the DIANA (Descriptive Intermediate Annotation for Ada) language has been proposed for this standard. No formal acceptance of DIANA is likely for several years. While DoD-funded

APSE development efforts are using DIANA as their intermediate form, most commercial Ada compilers are using something else (C language, P-code (Pascal derivative), etc.) and this will adversely affect tool portability.

5.3.2 Retargeting of Ada Programming Support Tools

Retargeting of an APSE means modifying it so that it can be used to develop Ada programs that execute on a different computer. An existing APSE may have to be retargeted for a particular computer as part of a system acquisition in order to satisfy storage requirements and other factors (e.g., computational speed, special peripherals, power requirements, weight, physical dimensions). Retargeting includes the following tasks:

- Redeveloping the target-instruction-set-dependent portions of the existing Ada compiler,
- Developing an assembler, loader, debugger, and other target-dependent tools for the new target computer (as circumstances dictate),
- Developing an Ada run-time environment for the target computer.

For the ALS, SofTech has estimated that it would cost between $1.0 - $1.5 million to complete minimum retargeting work and that it would take approximately 15 - 18 months to accomplish. Providing a symbolic debugger and a frequency analysis capability for the target machine would introduce an estimated additional expense of $400,000 and extend the development schedule 4 to 6 months. Telesoft has quoted retargeting of its compiler and loader at $300,000 and 6 months of development time. These software vendors have already amassed substantial backlogs of retargeting contracts.

Even after a software vendor such as SofTech completes its retargeting contract, an ESD program office can expect to incur additional retargeting-related expenses for such things as the following:

- Benchmark testing of the compiler and its object code,
- Compiler validation,
- Testing of tools for capacity, reliability, and response time,
- Maintenance of compiler and other tools and documentation,

- o Configuration management, problem report systems, and user assistance.

While the ALS development phase is being completed, the Army has released an interim (incomplete) version of ALS in October, 1983 to American companies for use in rehosting and retargeting to other computers. It is hoped that this will make the ALS available on a number of different mainframes and targeted to a number of mini- and micro-computers used in military applications. Some of these companies may seek funding for their work from a military contract.

5.3.3 Rehosting Ada Programming Support Tools

"Rehosting" of support tools means "moving from one host computer to another." In the next several years, there may be several instances in which APSEs will be rehosted, but these cases will be the exception rather than the rule. Rehosting necessitates:

- o Developing a new KAPSE.
- o Retargeting the target-dependent MAPSE tools, including the assembler, linker, and loader (as circumstances dictate), the host-instruction-set-dependent portions of the Ada compiler and possibly certain performance monitoring tools (like the debugger and the frequency analyzer).
- o Recompiling the complete MAPSE for use on the new host computer (after retargeting of the compiler to the new host computer). It should not be necessary to redesign the front end of a rehosted compiler.

SofTech, Inc. estimates that redevelopment of the ALS KAPSE would cost $1.5 million and require 18 months of redevelopment time. Redevelopment of the Ada compiler's code generator, and the assembler, linker, and loader has been estimated to cost an additional $1.0 million and would require 16 months to accomplish.

5.3.4 Validation of Ada Programming Support Tools

5.3.4.1 Compiler Validation. DoD policy requires that every new and/or retargeted Ada compiler used in acquisition programs shall be fully validated by the Ada Joint Program Office and that this validation process shall be repeated thereafter at 1-year intervals. Formal validation by DoD confirms that a compiler implements the full Ada language, to the extent that passing validation tests makes this confirmation possible. Experience has shown that even after validation, compilation problems still exist. For example, the

NYU-Ed compiler was experiencing an average of one new problem per day several months after it was validated by AJPO. More importantly, validation does not confirm that an Ada compiler will be effective during system acquisition. One can anticipate, for example, that new compilers may compile slowly and produce object code that is not sufficiently efficient for weapon system applications (e.g., with respect to object-code execution speed). Nevertheless, the Ada Compiler Validation Capability (ACVC) is widely regarded as being much more rigorous and thorough than any other comparable set of tests for some other high order language. It comprises about 2,000 test programs which average 200 source statements in length. Every one of these tests must be passed successfully in order for a compiler to be validated by AJPO. Of particular interest to the program manager, the ACVC tests do not stress the compiler in ways that might be applicable to large scale software developments; for example, the ACVC test programs, being relatively small, do not saturate symbol tables and library files used by the Ada compilers. For this reason, the program manager might wish to consider conducting some additional "heavy load" tests, in addition to benchmark testing to measure object code efficiency, before electing to use an Ada compiler in an acquisition program. Self-compiling and availability of support tools written in a given Ada compiler can add confidence to that compiler's maturity, but there is nothing that will replace the experience of a wide range of users in determining the reliability of a compiler.

Some AJPO policies associated with compiler validation are controversial. When new releases of compilers are made available by software vendors, AJPO expects DoD contractors to maintain currency relative to the latest validated releases. However, many DoD contractors would prefer to establish a compiler "baseline" early into an acquisition program, and avoid switching from one version of a compiler to another thereafter. Contractors are apprehensive about the uncertainties associated with re-validation of compilers. It is not entirely clear how a contractor should react midway into project in the event that a previously-validated compiler fails to pass the annual ACVS testing. As an illustration, a compiler may not undergo any changes whatsoever and still fail to pass re-validation because new problems, previously undetected, are surfaced by recent extensions to the ACVS test set. Additionally, some contractors believe that compliance with the complete Ada language standard isn't possible. These contractors are using cross compilers for embedded computers, such as those employed in Cruise Missile applications, and it isn't feasible to demonstrate adherence to the text I/O portions of the Ada language standard because printers are not included in their embedded computer configurations.

5.3.4.2 Validation of Ada Support Tools Other Than the Compiler. Whenever an APSE is either developed from scratch (e.g., ALS and AIE) or else rehosted and/or retargeted, it will require a certain amount of time to mature once it becomes available, and extensive acceptance tests should be required. At this time, there are no formal standards or AJPO-sanctioned validation suites for validating APSE software other than Ada compilers, and this situation is likely to continue. However, the Army procurement agency responsible for the ALS acquisition has prepared some tests of its own that might be used as a guide by ESD program managers confronted with rehosting/retargeting problems.

5.4 APSE SELECTION, ACQUISITION AND MAINTENANCE

5.4.1 Selection

5.4.1.1 Air Force Policy. The AFSC Ada Introduction Plan does not constrain System Program Offices and contractors tc using a single APSE. ESD contractors will be allowed to take advantage of a number of environments developed by both DoD and industry. This policy will allow ESD program managers to specify support software requirements and to select the most cost-effective APSE to meet their requirements.

Policy will be developed that will require APSEs and support tools used in Air Force developments to be written in Ada and to conform to interface standards* currently being developed by the KAPSE Interface Team (KIT). This policy will be developed with the availability of the interface standards and a sufficient number of Ada compilers to enable development of APSEs on a wide range of host computers. In the near term, in order to enable use of Ada as soon as possible, these requirements will not be levied.

5.4.1.2 Tool Choices Currently Available. In addition to the "accelerated maturity" and duplicate development strategies (discussed in section 3.5), several other strategies are relevant to the tool choice that the ESD program manager might wish to consider, all of which have their drawbacks. The following list is considered to be representative rather than all-inclusive.

- o Use a commercial Ada-subset compiler and associated tools as an interim measure until a full Ada compiler and companion tools become available. Several Ada-subset compilers are

*The Common APSE Interface Set (CAIS)

currently available for target computers that may be of interest to ESD Program Managers. Inasmuch as Telesoft has already sold more than 350 Ada-subset compilers, primarily targeted for the VAX and the MC68000 computers, it would appear that the subset approach is very popular. The use of an Ada-subset compiler in a DoD acquisition project would constitute a violation of draft DoD Directive 5000.31, but might be permissible if there is a firm commitment to shift to a fully validated Ada compiler prior to delivery of a system to the government (and there is a fallback HOL position in case the Ada-compiler vendor fails to deliver on schedule).

- Use a fully-validated commercially-funded compiler and associated tools such as the ROLM Ada Work Center. A few DoD contractors have adopted this approach. The commercially available compilers will not necessarily generate code for appropriate target machines and therefore may need to be retargeted. Most of the commercial Ada compilers do not utilize the DIANA intermediate form and this will tend to inhibit the introduction of special tools borrowed from other projects. The extent of maintenance support for tools other than compilers should be evaluated very carefully.

- Use the ALS or AIE. The Navy and the Army have both announced their intentions to use ALS as their standard APSE when it becomes fully available. Right now, the risks with this approach include schedule availability, compilation speeds, and reliability. Retargeting the Ada compiler and developing other necessary target support software may be a major consideration. It may be possible to design and develop Ada applications software in parallel with a retargeting effort, accomplishing initial checkout on an emulator or simulator or on a host computer. However, retargeting entails considerable risk because object code from an immature compiler is apt to be unreliable, the software vendor (due to work backlogs) may not be in a position to begin retargeting when desired, and retargeting in itself is a significant undertaking.

- Design the system in Ada-based PDL, code the software in some other approved HOL (e.g., JOVIAL), and then convert to Ada coding when one of the government-funded compilers and other APSE software becomes available, is validated, and judged to be mature. The use of an Ada-based PDL will mitigate, but not eliminate, the problems of reimplementing in Ada.

- Use a commercial Ada compiler that employs Pascal or C-code as an intermediate form (none of the commercial compilers with these intermediate forms are currently validated). The Pascal or C generated by the Ada compiler will serve as input to an existing mature Pascal or C compiler, respectively, which will produce the object code for the target machine. Existing operating systems, run-time environments and tools for support of Pascal or C program integration and testing would be used temporarily. Eventually, all of the Ada source code, initially developed in this way, would be transferred over to one of the DoD funded APSEs (AIE or ALS). This approach requires programmers to contend with more than one high order language and is considered to be unwieldy and time consuming. Nevertheless contractors occasionally are proposing this sort of thing.

The program manager may choose to forestall the Ada language decision temporarily by adopting the following approach:

- Require the use of an Ada-based PDL.
- Require that some amount of prototype software development be accomplished using Ada early in the acquisition program.
- Require various types of stress testing (maximum program size, maximum symbol table use, etc.) of the Ada compilers that are potentially of interest.
- Benchmark the object-code efficiency of Ada compilers versus that of other approved-HOL compilers. Also measure compilation speeds of candidate Ada compilers.
- Defer the language decision until partway into the system acquisition.
- Require the contractor to provide a backup plan in case an Ada compiler is not delivered on time, proves to be unreliable, and/or generates unacceptably inefficient object code.

This approach is similar to, but not the same as, the "Duplicate Development" strategy described in section 4.2.

5.4.2 Acquisition of an APSE

The Air Force[7] does not intend to provide APSEs as Government Furnished Property or Equipment (GFP/GFE) to ESD contractors. This

approach is consistent with current Air Force acquisition practices. As a result, the ESD contractor must either develop or purchase Ada programming tools from software vendors.

5.4.3 Maintenance of an APSE

Unlike the Army and the Navy, the Air Force does not intend to maintain APSEs used by DoD contractors. This approach will curb the need to establish an organic AFSC Product Division with Ada tool support capabilities. As part of the contractor's proposal, there should be a task that provides for the maintenance of Ada Programming Support Tools.

5.5 COST ESTIMATION AND SCHEDULING

During the next couple of years, ESD program managers will experience considerable difficulty trying to evaluate the reasonableness of software cost estimates and development schedules when Ada is to be the implementation language for an acquisition project. In the short run, using Ada is likely to cost more and take longer than using any other DoD-approved HOL.

- First, very little information is available about the relative verbosity of Ada versus another language, such as JOVIAL. Ada might require more lines of source code than JOVIAL to accomplish the same function (e.g., a Kalman Filter), but there is no empirical data available to support this contention. The sizing estimates used as inputs to software cost models are usually based on previous experience in similar systems with a common programming language; in the case of Ada, this set of knowledge simply does not currently exist.

- Second, programmer productivity data, expressed in terms of lines of Ada code per man month, generally is not available. What little data is known tends to exaggerate the claims of programmer productivity because it is based on the experience of select groups of extraordinarily talented people working on relatively small projects without being subjected to MIL-STDs for software documentation. This data is not considered to be representative of what might be expected when many programmers of less sophisticated qualifications are turned loose on a large contract.

- Third, software cost models (e.g., PRICE-S, Putnam, Cocomo, BOEING) will have to be recalibrated for Ada implementations. Recent experience with Ada indicates that an

unusually large fraction of the overall software development effort is consumed during preliminary design with Ada while a correspondingly smaller fraction of effort is expended during integration testing. These observations have not yet been taken into consideration by the various cost models.

- Finally, the costs for using Ada-based PDLs are unknown.

5.6 PROJECT AND CONFIGURATION MANAGEMENT

The overall introduction of Ada in DoD acquisitions is more than just adopting a new programming language; the standardization of the support environment is significant in its own right. APSE standardization may well have more effect on the ESD program manager than language standardization. In the past, the contractor's configuration control, scheduling, bug tracking, and other systems were largely hidden from or inaccessible to the program manager; they sometimes ran on whatever mainframe the contractor's accounting department used or else were manual. The information captured by these systems tended to reach the program manager through contractor status reports and briefings, often late, highly condensed, and possibly slanted. With a standard APSE, the program manager can learn to be a user and direct the contractor to use standard project and configuration management tools. The program manager can then inspect management data, source and documentation text, test results, etc., to keep as close tabs on the contractor as desired. The program manager should be aware of the possibility of better visibility into status while writing the RFP and negotiating the contract. Many details of remote access (telephone link to contractor's host computer) and data availability will need to be worked out. Even if he doesn't wish to use remote access, an informed program manager will understand what types of information an APSE makes possible and can request the data and reports he wants.

The MAPSE includes mechanisms and tools intended to support configuration management. For example, an object file stored in the MAPSE file system is linked to the source file from which it was produced by the compiler. If that source file is changed, the MAPSE will either create a new "revision" of it with the changes or erase the outdated object file. In either case, the exact source used to create an object file is always available.

It is expected that the MAPSE user will employ the facilities of the file system and canned command procedures to create a configuration management system of his own. This is to say, both the contractor and the program office should expect to create their

own configuration management tools, because those contained in the MAPSE are less than complete. An ESD program manager might wish to have a set of tools for report generation (which will access the contractor data base via telephone linkages to determine what documentation has been written, which source code has been compiled successfully, and what tests have been run). Because the MAPSE is standard, the program management personnel knowledgeable about the MAPSE should be able to understand and evaluate the contractor's configuration management system. The program manager can request that the configuration management plan be included in the proposal and can check that it is being followed by accessing the APSE. Conversely, the draft Computer Resources Integrated Support Plan (CRISP) included in the RFP package submitted to the bidders should reference any tools that the program manager wishes to have made available for use in configuration management.

The program manager will be able to "trust" the MAPSE in taking delivery of software from it. For example, in past practice, delivery was often of a source-only tape, which would be laboriously compiled and tested. Because the MAPSE can "guarantee" that a given object file is the result of compiling a given source file or that a given test has been run successfully, delivery can instead be a tape of the most up-to-date revisions of all files.

There is one intrinsic danger in using configuration management tools in the types of computers that are currently being used to host Ada Programming Support tools. A single host computer simply may not be big and fast enough to provide the level of support necessary for a large number of programmers involved in the development of a complex C^3I system. For example, the ALS will support at most 15 users concurrently on the VAX 11/780 computer. As a result, multiple VAXs will potentially be needed for a large acquisition project. However, the configuration management capabilities inherently available in the MAPSEs today are designed to run on a software development facility comprising a single host computer, and it is unclear what sort of procedures and additional software capabilities may be needed to support configuration management in a distributed environment (i.e., in more than one host computer). The contractor's approach for configuration management on a large system is especially important and should be considered among the various evaluation criteria.

5.7 TRAINING

Personnel training problems are expected to be a major area of concern to ESD program managers during the next several years, both because the reservoir of programmers within the defense contractor

community who are knowledgeable in Ada is smaller than what DoD will require, and because experience with Ada in ESD itself is almost nonexistent. The transition towards mandatory use of Ada is moving faster than expected. Ada is not easy to learn, and the training courses commercially available are not always effective because they sometimes are focused on language features and not on the concepts that support the language features themselves.

The typical programmer will experience considerable difficulty trying to learn Ada and will have to adopt an entirely new way of thinking about how to accomplish software development. Ada is not just another programming language. Most programmers in the defense industry have backgrounds in assembly language and FORTRAN, and these people will experience the greatest difficulty with Ada. They will tend to apply Ada in the manner of these earlier languages, and thereby fail to exploit the full capabilities of Ada. Programmers with some degree of Pascal language background will acquire Ada skills more rapidly because the Ada language is derived from Pascal.

Ada is difficult to learn because the terminology and basic concepts are difficult. Some consulting firms that specialize in teaching Ada have found that a significant number of programmers are intellectually incapable of understanding (or else unwilling to learn) "packaging," "data abstraction," "generic program units," and "concurrent tasking." Another difficulty is that Ada cannot be taught simply in terms of syntax. Software engineering principles (structured programming, functional decomposition, information hiding, data abstraction, top-down design) need to be stressed along with the Ada language. This extra schooling takes time.

The time required to teach someone Ada will vary according to job classification and general learning ability. Project managers should be encouraged to identify someone as their "Ada expert," a person who will be responsible for assisting other programmers on the project. Recent experience has shown that these Ada experts sometimes require as much as 3 months of intensive training but it varies from one individual to the next. The average Ada programmer needs about 1 month of training in order to become self-sufficient while writing code.

Training in Ada should occur early in an acquisition program because the use of Ada will impact the very first design decisions and perhaps even affect the requirements analysis phase. It should include designing, coding, and testing in Ada.

Unfortunately, many corporate project managers are underestimating the magnitude of the training problem and, therefore, are unwilling to devote the resources to train people properly. In the

past, programmers could take a basic course in FORTRAN for 1 week and then begin programming. Consulting houses are frustrated because they are given only 1 week to teach Ada to someone, and this isn't enough time to even cover the basic principles. Besides needing more time, the consulting houses should use teaching methods that replace the traditional formal lectures. These techniques include hands-on experience with computers outfitted with Ada compilers, computer assisted training aids, practical homework problems, very low student-teacher ratios, and frequent examinations.

Even with proper training, it will take several years for contractors to become familiar with applying the Ada language in C^3I applications. Up until now, most of the contractors have only experimented with Ada in small R&D projects involving 10 people or less.

5.8 CURRENT PROBLEMS USING ADA AS AN IMPLEMENTATION LANGUAGE

Recent experiences[10] with Ada as an implementation language have evidenced the following problem areas:

- Execution speed of Ada-compiled code (is not fast enough); (The compiler vendors are working hard to devise new techniques for code optimization).
- Memory storage requirements for Ada-compiled code (is too large).
- Difficulties with interfacing with off-the-shelf software packages such as Data Base Management Systems, Graphic Display Systems, etc.
- Some of the existing Ada compilers and run time environments do not provide features for executing tasks at more than one priority level.
- Ada's tasking features do not respond quickly enough to real-time stimuli.
- File I/O is inappropriate for non-real time applications.
- Certain Ada implementations entail the use of dynamic memory allocation (i.e., temporary work space) and Ada run time environments do not always provide features for (1) determining how much dynamic memory still remains unused and (2) automatic "garbage collection" to recover work space in the

event that the dynamic memory region is fully depleted. Dynamic memory allocation and de-allocation may prove to be overly time-consuming in certain applications.

- o Use of address specifications in interrupt handlers require that an address specification be static. This limitation precludes any dynamic reassignment of tasks to interrupts that is a requirement in some systems.

- o Existing compiler implementations do not currently support the use of a rendezvous between processors in a distributed processing environment.

Some of these current problems are likely to disappear over time as existing compilers incorporate improved optimization routines, and are extended to include new features not presently available.

LIST OF REFERENCES

1. US DoD, Reference Manual for the Ada Programming Language, MIL-STD-1815A, 17 February 1983.

2. J. Buxton, Department of Defense Requirements for Ada Programming Support Environments "STONEMAN," DoD, February 1980.

3. US DoD, Nebula Instruction Set Architecture, MIL-STD-1862, 3 January 1983.

4. US DoD, Management of Computer Resources in Major Defense Systems, DoDD 5000.29, 26 April 1976.

5. US DoD, Interim List of DoD Approved High Order Programming Languages (HOL), DoDI 5000.31, 24 November 1976.

6. USAF, Computer Programming Procedure for Managing Automated Data Processing System, AFSC Supplement 1, AFR 300-10, 2 September 1980.

7. AFSC, Ada Introduction Plan, September 1983.

8. US DoD, Defense System Software Development, MIL-STD-SDS, 15 April 1982 (draft).

9. USAF, Configuration Management Practices for Systems, Equipment, Munitions, and Computer Programs, MIL-STD-483A, 15 April 1982.

10. Conn, H., "Ada Capability Study, Final Report," General Dynamics Corporation, Fort Worth, Texas, sponsored under US Army CECOM Contract No. DAAK 80-81-C-0108.

BIBLIOGRAPHY

Bjerstedt, W. R., and J. B. Glore, Software Acquisition and Management Guidebook: Statement of Work Preparation, ESD-TR-77-16, Electronic Systems Division, AFSC, Hanscom AFB, MA, January 1977, AD A035 924.

Bolen, N. E., An Air Force Guide to Contracting for Software Acquisition, ESD-TR-75-365, Electronic Systems Division, AFSC, Hanscom AFB, MA, January 1976, AD A020 444.

Booch, G., Software Engineering with Ada, Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA (1983).

Braun, C., "Understand Ada's Capabilities to Promote Software Savings," Electronic Daily News, April 28, 1983, pp. 181-187.

_____,"Ada Training Considerations," SofTech Report sponsored under US Army CECOM Contract No. DAAK 80-81-C-0187.

Bratman, H., and M. C. Finfer, Software Acquisition Management Guidebook: Verification, ESD-TR-77-263, System Development Corporation, August 1977, AD A048 577.

Ehrenfried, D., "Feasibility Assessment of JOVIAL to Ada Translation," AFWAL/AAAF-2.

Finfer, M., Software Acquisition Management Guidebook: Cost Estimation and Measurement, ESD-TR-78-140, System Development Corporation, March 1978, AD A055 574.

Glore, J. B., Software Acquisition Management Guidebook: Life Cycle Events, ESD-TR-77-22, Electronic Systems Division, AFSC, Hanscom AFB, MA, March 1977, AD A037 115.

Gold, H. I., and G. Neil, Software Acquisition Management Guidebook: Software Quality Assurance, ESD-TR-77-255, System Development Corporation, August 1977, AD A047 318.

Hagan, S. R., and C. W. Knight, An Air Force Guide for Monitoring and Reporting Software Development Status, ESD-TR-75-85, Electronic Systems Division, AFSC, Hanscom AFB, MA, September 1975, AD A016 488.

BIBLIOGRAPHY (Continued)

Mish, R. K., Software Acquisition Management Guidebook: Series Overview, ESD-TR-78-141, System Development Corporation, March 1978, AD A055 575.

Neil, G., Software Acquisition Management Guidebook: Reviews and Audits, ESD-TR-78-117, System Development Corporation, November 1977, AD A052 567.

Nissen, J. C. D., P. Wallis, B. A. Wichmann et al, "Ada-Europe Guidelines for the Portability of Ada Programs," National Physical Laboratory Report, DNACS 52/81, November 1981.

Peterson, D. R., Software Acquisition Management Guidebook: Software Development and Maintenance Facilities, ESD-TR-77-130, Electronic Systems Division, AFSC, Hanscom AFB, MA, April 1977, AD A038 234.

Schoeffel, W. L., An Air Force Guide to Software Documentation Requirements, ESD-TR-78-159, Electronic Systems Division, AFSC, Hanscom AFB, MA, June 1976, AD A059 202.

Searle, L. V., An Air Force Guide to Computer Program Configuration Management, ESD-TR-77-254, System Development Corporation, August 1977, AD A047 308.

_____, An Air Force Guide to the Computer Program Development Specification, ESD-TR-78-139, System Development Corporation, November 1977, AD A055 573.

Skrukrud, A. M., and N. E. Willmurth, Software Acquisition Management Guidebook: Validation and Certification, ESD-TR-77-326, System Development Corporation, August 1977, AD 053 039.

Skrukrud, A. M., and J. R. Stanfield, Software Acquisition Management Guidebook: Software Maintenance, ESD-TR-77-327, System Development Corporation, October 1977, AD A053 040.

Thall, R. M., "The KAPSE for the Ada Language System," Proceedings of the AdaTEC Society, Arlington, Virginia Conference, October 6-8, 1982.

BIBLIOGRAPHY (Concluded)

USAF, Configuration Management Practices for Systems, Equipment, Munitions, and Computer Programs, MIL-STD-483, 21 March 1979.

USAF, Engineering Management, MIL-STD-499A, 1 May 1974.

USAF, Technical Reviews and Audits for Systems, Equipment, and Computer Programs, MIL-STD-1521, 21 December 1981.

USAF, Statement of Operational Need (SON), AFR 57-1, 12 June 1979.

USAF, Source Selection Policy and Procedures, AFR 70-15, 16 April 1976.

USAF, Test and Evaluation, AFR 80-14, 12 September 1980.

USAF, Acquisition Program Management, AFR 800-2, 13 August 1982.

USAF, Computer Programming Languages, AFR 300-10, 15 December 1976.

USAF, Engineering for Defense Systems, AFR 800-3, 17 June 1977.

USAF, Acquisition and Support Procedures for Computer Resources in Systems, AFR 800-14, Vol. II, 26 September 1975.

US DoD, Configuration Control-Engineering Changes, Deviations and Waivers, MIL-STD-480A, 29 December 1978.

US DoD, Work Breakdown Structures for Defense Material Items, MIL-STD-881A, 25 April 1975.

US Navy, Weapon System Software Development, MIL-STD-1679, 1 December 1978.

APPENDIX

DRAFT VERSION OF REVISED DODD 5000.31

The following is a draft version of DoDD 5000.31 that was expected to be formally issued as a directive in September, 1983.

THE UNDER SECRETARY OF DEFENSE
WASHINGTON, D.C. 20301

RESEARCH AND ENGINEERING

10 JUN 1983

MEMORANDUM FOR SECRETARIES OF THE MILITARY DEPARTMENTS
CHAIRMAN OF THE JOINT CHIEFS OF STAFF
UNDER SECRETARY OF DEFENSE FOR POLICY
ASSISTANT SECRETARY OF DEFENSE, COMPTROLLER
GENERAL COUNSEL, DEPARTMENT OF DEFENSE
INSPECTOR GENERAL, DEPARTMENT OF DEFENSE
DIRECTOR, DEFENSE INTELLIGENCE AGENCY
DIRECTOR, DEFENSE NUCLEAR AGENCY
DIRECTOR, NATIONAL SECURITY AGENCY

SUBJECT: Interim DoD Policy on Computer Programming Languages

I have reviewed the attached draft revision of DoD Directive 5000.31 which is being circulated for final coordination. Pending formal coordination and publication of this directive, I request that it be implemented immediately as DoD policy concerning the use of high order languages in mission-critical Defense applications. Because of the increasing importance of software in Defense systems and due to intense congressional interest, we are elevating this policy from Instruction to Directive level.

Ada® (ANSI/MIL-STD-1815A-1983) is approved for use consistent with the introduction plans of the individual components and the validation requirements of the Ada Joint Program Office. The Ada programming language shall become the single, common computer programming language for Defense mission-critical applications. Effective 1 January 1984 for programs entering Advanced Development and 1 July 1984 for programs entering Full-Scale Engineering Development, Ada shall be the programming language. "Mission-critical" applications are those exempted from the Brooks Act by 10 U.S.C. 2315, the Warner Amendment to the FY 1982 Defense Authorization Act. My memorandum of 4 March 1983 transmitted further guidance relative to this legislative action. Other programs are encouraged to use Ada as soon as and whenever possible.

The introduction plans required in Section F.2.e Draft DoDD 5000.31 should be prepared and coordinated with Director, Ada Joint Program Office, not later than 15 August 1983 so that they can present them for my approval by October 1, 1983.

Attachment

NUMBER

Department of Defense Directive

SUBJECT: Computer Programming Language Policy

References: (a) DoD Instruction 5000.31, "Interim List of DoD Approved High Order Programming Languages," November 24, 1976 (hereby canceled)
(b) DoD Directive 5000.29, "Management of Computer Resources in Major Defense Systems," April 26, 1976 [Under Revision]

A. REISSUANCE AND PURPOSE

This Directive replaces reference (a) and specifies the computer programming languages authorized for use in Defense mission critical applications. It provides both policy and procedures for the management and control of these programming languages and their associated programming support environments. It authorizes the publication of DoD 5000.31-M, a manual which prescribes guidance and procedures governing the management of computer programming languages and programming support environments.

B. APPLICABILITY AND SCOPE

1. This Directive applies to the Office of the Secretary of Defense, the Military Departments, the Organization of the Joint Chiefs of Staff, and the Defense Agencies (hereinafter collectively referred to as "DoD Components").

2. This Directive need not be applied retroactively to DoD systems for which a formal language commitment has been made in compliance with reference (a) before the effective date of this revision.

C. DEFINITIONS

Software engineering terms used in this Directive are defined in IEEE Standard 729-1983, "IEEE Standard Glossary of Software Engineering Terminology" or in reference (b).

D. POLICY

1. The number of distinct programming languages used within DoD must be minimized.

2. All software developed for mission critical systems shall be developed and written in a programming language authorized by this Directive unless a waiver is obtained in accordance with

this Directive. Waivers are required for the use of extensions or enhancements of authorized languages as well as for the use of unauthorized languages. Programming languages authorized for use in mission critical applications are set forth in Enclosure 1.

3. The configuration of programming languages approved for use in Defense systems must be controlled in accordance with appropriate configuration management directives.

4. The Ada[1] programming language shall become the single, common, computer programming language for Defense mission-critical applications. Effective January 1, 1984 for programs entering Advanced Development and July 1, 1984 for programs entering Full-Scale Engineering Development, Ada shall be the programming language. Only compilers which have been validated by the Ada Joint Program Office shall be used for software to be delivered to or maintained by the government.

E. PROCEDURES

1. Control Agent.

Each DoD-approved programming language is assigned to a designated control agency. The control agency is responsible for maintaining a single standard definition of the assigned language and for making this definition document

---

[1] Ada is a registered trademark of the Department of Defense (Ada Joint Program Office) OUSDRE (R&AT)

available as a Federal, DoD, military or adopted Industry Standard. For those languages, the definitions of which are industry standards controlled outside DoD, a DoD component is assigned for the required additional responsibilities and to represent DoD to the controlling organization. The control agency shall also be responsible for providing configuration control of the assigned language, and for gathering data and disseminating appropriate information regarding use of the language, its compilers and associated tools. The designated control agency for each authorized DoD language is set forth in Enclosure 1.

2. Waivers.

a. A waiver need not be obtained for use of commercially available software for use in routine business or administrative applications (non-mission critical), or for use of an application-oriented (including problem-oriented) language, or for commercially available, off-the-shelf software if no modification of that software is anticipated over the life cycle.

b. Waivers from the use of an approved programming language may be granted by the DoD Component only on a specific system or subsystem basis. The costs and risks associated with language proliferation must be weighed against the waiver benefits accruing to the intended (sub)system. A summary of this analysis will be forwarded with each waiver granted to the Defense Computer Resources Board which will review Component-granted waivers and, within 30 days, may reverse or otherwise disapprove the waiver.

3. Addition or Deletion of Approved Languages.

a. A Component may nominate a language for inclusion on the list of approved languages by submitting to the Defense Computer Resources Board a document which: (i) describes the language, (ii) provides a detailed specification of the language, (iii) provides detailed rationale for adopting the language as a DoD-approved standard, (iv) sets forth an economic analysis of the impact of the language for its expected life-cycle, (v) includes a detailed plan for implementation and life cycle support of the language, and (vi) identifies the DoD Component that will accept designation as Control Agency for the language.

b. For deletions, the DoD Component nominating a language for removal from the list of approved languages will submit to the Defense Computer Resources Board a document which presents the rationale for deleting the language, including an impact analysis and a detailed plan for transitioning current programs which use the language to another of the approved languages.

c. After receipt of the nomination document, the Defense Computer Resources Board will distribute the nominating document, request dissenting reports, and schedule a briefing by the nominating DoD Component.

d. The Defense Computer Resources Board will act upon the nomination within 60 days of the briefing by the nominating DoD Component.

e. Appeals to decisions regarding the acceptance or rejection of nominations may be submitted via the appropriate chain of command to the Office of the Under Secretary of Defense for Research and Engineering.

F. RESPONSIBILITIES

1. The Deputy Under Secretary of Defense (Research and Advanced Technology) shall be the DoD Senior Official for all activities related to this Directive and shall:

a. Oversee, coordinate and implement the policies and procedures of this Directive;

b. Advise the Office of the Secretary of Defense on all matters related to this Directive; and

c. Be the authority for adding or deleting computer languages to or from the authorized list.

2. The Heads of the DoD Components shall:

a. Implement this Directive within their respective organizations.

b. Institute procedures and coordinate with the Defense Computer Resources Board.

c. Designate a computer language Waiver Control Officer(s) who shall have responsibility for implementation of the waiver provisions and procedures of this Directive.

d. Designate coordinating offices for each approved language which is of application interest, if their Component has not been assigned as control agent for it.

e. Prepare and maintain a plan for the introduction of Ada, and the phase-out of other languages from consideration for new programs, which shall be coordinated with the Ada Joint Program Office for incorporation into the Ada Program Management Plan.

f. Ensure that use of the Ada programming language by their Component is consistent with the Component's Ada introduction plan, that use of the Ada language is permitted on an unrestricted basis, and that the use of the Ada programming language is actively encouraged.

g. Designate a language control officer for each language for which the Component is assigned as the Language Control Agent, such officer to have the authority and responsibility for proper support for all associated language control activities. Each language control officer is authorized to issue, upon approval of the Defense Computer Resources Board, a revised version of the assigned language in order to provide modifications and improvements in satisfaction of validated requirements, or to resolve ambiguities in the defining document. Such revised versions should not be issued more frequently than once each year.

G. EFFECTIVE DATE AND IMPLEMENTATION

This Directive is effective immediately. Components shall forward five copies of implementing documents to the Under Secretary of Defense, Research and Engineering, prior to October 1, 1983.

Enclosure - 1

1. Approved Computer Programming Languages

APPROVED COMPUTER PROGRAMMING LANGUAGES

The DoD-approved Computer Programming Languages follow. In each case, the applicable standard is defined by the most recent official version of the designated document.

1. Ada; "Ada Programming Language", ANSI/MIL-STD-1815A, February, 1983.

   Control Agency: American National Standards Institute

   DoD Control Agency: Ada Joint Program Office

2. CMS-2M; "CMS-2M Computer Program Performance Specifications," NAVSEA 0967LP-598-2210, April 1982, and CMS-2Y; "CMS-2Y Programmers Reference Manuals M-5049, M-5044, April 15, 1981.

   Control Agency: Department of the Navy

3. JOVIAL (J73); Military Standard MIL-STD-1589B (USAF), 6 June 1980.

   Control Agency: Department of the Air Force

4. FORTRAN; "American National Standard X3.9-1978"

   Control Agency: American National Standards Institute

   DoD Control Agency: Department of the Air Force

5. COBOL; "American National Standard X3.23-1974"

   Control Agency: American National Standards Institute

   DoD Control Agency: Department of the Air Force

NOTE: COBOL is to be used only for business or administrative applications. Consideration should be given to use of Ada where appropriate for machine independence.

6. Special Language for Automatic Test application: ATLAS and C/ATLAS; "IEEE Standard ATLAS Test Language," ANSI/IEEE Standard 416-1980; "IEEE Standard C/ATLAS Test Language," IEEE Standard 716-1982; and "IEEE Standard C/ATLAS SYNTAX," IEEE Standard 717-1982.

   Control Agency: Institute of Electrical and Electronic Engineers

   DoD Control Agency: Department of the Navy

End of List

GLOSSARY

Acronyms and Abbreviations

ACVO	Ada Compiler Validation Organization
AFATL	Air Force Armament Test Laboratory
AFSARC	Air Force Systems Acquisition Review Council
AFSC	Air Force Systems Command
AIE	Ada Integrated Environment
AJPO	Ada Joint Program Office
ALS	Ada Language System
APSE	Ada Programming Support Environment
ATLAS	Automatic Test Language for Avionic Systems
C^3I	Command, Control, Communications, and Intelligence
COBOL	Common Business Oriented Language
CORADCOM	Army Communications Research and Development Command
CPDP	Computer Program Development Plan
DBMS	Data Base Management System
DIANA	Description Intermediate Attributed Notation for Ada
DSARC	Defense System Acquisition Review Council of OSD
ESD	Electronic Systems Division
FORTRAN	Formula Translator
GFE	Government Furnished Equipment
GFP	Government Furnished Product (or Property)
HOL	High Order Language
IFPP	Instruction for Preparation of Proposals
JOVIAL	Jules Own Version of International Algorithmic Language
KAPSE	Kernel Ada Programming Support Environment
KIT	KAPSE Interface Team
MAPSE	Minimal Ada Programming Support Environment
RADC	Rome Air Development Center
RFP	Request for Proposal
SOW	Statement of Work

A Level System Specification. A specification written by the government, which states the technical and mission requirements for a system, allocates the requirements to functional areas, defines interfaces between the functional areas, and specifies constraints on the design of the system.

Accelerated Tool Development. Activity undertaken over a relatively short period of time to mature Ada programming support software (tools) after their development has been completed, by detecting residual errors during trial applications and stress testing, correcting the detected errors, and in general improving the performance and usability of the software.

Ada Program Units. The four basic kinds of building blocks of Ada programs. Each program unit has a specification part that defines how it can be used, and a body part that implements the specification. (See Subprogram, Package, Generic, and Task)

Air Force Language Control Facility. An Air Force office at the Wright Patterson Air Force Base responsible for the validation and maintenance of JOVIAL compilers.

Application-Oriented Language. A computer programming language with constructs and other features customized toward a specific application (e.g., LISP for pattern recognition, ATLAS for equipment diagnostics, and Ada for embedded systems).

Army Military Computer Family. A family of compatible machines, based on a standard 32-bit instruction set architecture known as Nebula, specified in MIL-STD-1862.

Assembler. A computer program that translates another computer program written in symbolic machine language into the machine code of a particular machine. The translation is often a one-to-one transformation.

Case. A statement of a computer programming high order language, which transfers control to one of several possible alternative sequences of statements, depending on the value of its control expression.

Commercially Funded Work. Work undertaken by industry that is not directly funded by the government.

Compiler-Time Performance. The time it takes a compiler to translate source language statements into machine code for a particular machine.

Compiler. A computer program that translates a high order language program into its relocatable assembly or machine code equivalent.

Configuration Management. The process of identifying and defining the configuration items in a system, controlling the release and change of these items throughout the system life cycle, recording and reporting the status of configuration items and change requests, and verifying the completeness and correctness of configuration items.

Data Structure. A formalized representation of the ordering and accessibility relationships among data items without regard to their actual storage configuration.

Data Type. A class of data characterized by the members of the class and the operations that can be applied to them (e.g., integer, real, logical).

Defense Computer Resources Board. A DoD agency with responsibility for reviewing waivers granted relative to the use of Ada by the Army, Air Force, or Navy.

Design Methodology. An approach to creating a design, consisting of an ordered application of techniques, guidelines, and tools.

Design Verification. The act of reviewing, inspecting, testing, checking, and otherwise establishing that a design conforms to specified requirements (e.g., to confirm compliance of a Type B5 software development specification to an A level system specification) and confirming that the design implements the requirements in a feasible, efficient, and correct manner, in accordance with accepted good practice.

DIANA (Descriptive Intermediate Attributed Notation for Ada). A machine-independent representation of Ada source code used within compilers to link the front and back ends of the compiler, providing symbol-state information required for cross-compilation checking. The intermediate form is used by debuggers and, as circumstances dictate, may also be used by other programming support software. DIANA is suitable for interfacing with programming support tools.

DIANA is based upon TCOL-Ada (Carnegie-Mellon University) and AIDA (i.e., an Intermediate Description of Ada developed at Technische Universitaet Karlsruhe, Germany).

DoD Components. Geographically or otherwise separated organizational units of the Department of Defense.

Dynamic Analysis Tools. Programming support software that aids the evaluation of computer program execution (e.g., by providing snapshot, trace, breakpoint, stub, interface simulator, statement execution monitor, and timing analysis capabilities).

Environmental Simulator. Software and hardware test tools that generate and transmit real time signals at equipment interfaces (of the system under test), simulating environmental activity and responding in a dynamic manner to received signals generated by the computer programs being tested. An environmental simulator is typically used in the integration testing of computer programs.

Exception. An event that causes suspension of normal computer program execution.

File Administration. The act of providing and controlling access to files, directing their maintenance, and controlling the resources used.

Function. See Subprogram.

Functional Decomposition. A method of designing a system by organizing its requirements into components, which, when taken in their entirety, implement the required system functions.

Generic. An Ada subprogram or package that is a template or "macro" for defining other program units. Generics may not be directly executed. (See Instantiation)

High Order Language (HOL). A programming language providing relatively powerful features not found in machine or assembly languages (e.g., nested expressions, user defined data types).

Instantiation. The process of creating a specific definition of subprogram or package parts, by making a one-to-one substitution of an actual parameter value for every formal generic parameter.

Implementation Language. The computer program language used to generate source instructions.

Integrated Support Software. A set of computer programs used to create, edit, compile, and unit test application computer programs and manage this software development work; the set of computer programs are typically stored together for access by a user.

Lexical Analysis. The analysis of basic syntactic elements making up a computer program, (i.e., the analysis of the way numbers, alphabetic characters, comments, and punctuation marks are put together as a source language in accordance with predefined rules).

Life Cycle Software Support. Providing support to the development of a software product from the time the product is conceived to the time the product is no longer used, including phases associated with requirements specification, design, implementation, test, installation and checkout, operation and maintenance, and product retirement.

Linker. A computer program used to create one load module from one or more independently compiled modules by resolving cross-references among the object modules, and possibly by relocating elements.

Loader. A software routine that reads an object program into main storage prior to its execution.

Module. A logically separable part of a computer program that is discrete and identifiable with respect to compiling, combining with other units, and loading.

Modular Structure. The manner in which software is constructed using modules. Descriptions of modular structure focus on the interrelation of modules and the definition of effects achieved by each module.

Multi-Tasking. The concurrent execution of two or more tasks by a computer.

Nebula. An Army standard 32-bit instruction set architecture specified in MIL-STD-1862.

Object Code Efficiency. Measures of how object code performs its intended functions with respect to its utilization of computing resources (e.g., CPU time, memory).

Operating System. Software that controls the execution of programs on computer systems, providing services such as resource allocation, scheduling, input/output control, and data management.

Package. An Ada program unit consisting of a collection of entities, including data types, data objects, procedures, functions, tasks, and other packages which, in practice, are typically related. An Ada package as a whole cannot be invoked but the procedures, functions, and tasks it encapsulates can be individually made available to and invoked by other program units. A package helps facilitate abstraction in the structure of an Ada computer program by capturing lower level detailed entities not needed at higher levels. The specification and body of a package can be separately compiled. (See Ada Program Unit)

Performance Monitor. Computer programs used to capture data that measures the ability of modules, computer programs, or computer systems to perform their specified function within acceptable time constraints. For example, a performance monitor might measure response time and throughput.

Private Types. A data type whose structure and set of values have been clearly defined but are visible to external users of the type.

Procedure. See Subprogram.

Program Design Language. A language with constructs similar to a computer programming language used in conjunction with text and structured in an organized manner to present the "blueprint" for a computer program. There is no formal government specification of what a PDL must encompass but it is generally felt that a PDL should express a computer program's structure, control, logic, and algorithms; a PDL may also, either directly (using text) or indirectly (as a by-product of examining its sequenced statements), express a computer program's structure, variable and data names, data structure, and data flow between modules.

Programming Standards and Conventions. The rules and techniques applicable to the design, implementation, and documentation of computer programs (e.g., use of PDLs, hierarchical decomposition,

top-down design, incremental releases, structured code, and source listing preparation rules).

Programming Support Environment. An integrated collection of programming support software (e.g., a compiler, editor, loader and debugger) referred to as tools, which provide programming support capabilities throughout life cycle of a software acquisition.

Psuedo Code. A combination of programming language and natural language used for computer program design (e.g., as part of a PDL).

Real Time. Processing of data in a computer at rates that are faster than those required by processes performed outside the computer.

Rendezvous. The interaction that occurs between two tasks of an Ada computer program, when one task indicates to a second task that a transfer of information or some other form of communication is being requested, and the second task receives and accepts the request and responds accordingly.

Retargeting. The activity undertaken to make computer programs (including application software and programming support tools) available for use in conjunction with a computer other than that for which they were originally intended.

Run Time Environment. Software and other facilities used in conjunction with the execution of object code in a computer; it includes the software that facilitates I/O, tasking, exception handling, generics, and math functions (e.g., square root, cosine and sine).

Software Cost Model. A mathematical formulation or a procedure for estimating software costs using parametric estimation techniques or a model of the software acquisition process.

Software Maturity. The quality or state that software reaches when it has been fully developed, tested, improved, and made efficient and user friendly.

Source Instructions. The set of computer program instructions written by a computer programmer. Typically source instructions are written in a high order language and are translated to lower-level instructions (e.g., object code) via compilation processes.

Specification Section of an Ada Program Unit. The part of an Ada program unit that defines its constants, variables, types, and program units (tasks).

Static Analyzer. A software support computer program (tool) that aids in the evaluation of a computer program without executing the program (e.g., syntax checkers, cross-reference generators, and flowcharters).

Stub Generator. A software support computer program used to insert dummy software program units in place of operational units prior to the availability of the operational units. The dummy program units typically neither implement logic nor make calculations, but satisfy interface requirements, and can consume computer time and use memory space.

Subprogram. An Ada program unit that is either a procedure or a function. A procedure is the program unit (and the only program unit) that can be used as the main program. A procedure can also be invoked by a function, package, task, or another procedure to calculate one or more parameters and/or perform one or more set of actions. A function is embedded in an Ada statement (e.g., a control statement or arithmetic expression) and calculates a single parameter when the statement is invoked. (See Ada Program Unit)

Symbolic Debugger. A computer program (referred to as a programming support tool) used to support computer program testing. A symbolic debugger provides the capability to set breakpoints, printout variable values, and supply other testing related information based symbols of the source listing rather than absolute machine addresses.

Syntactic Analysis. The examination and evaluation of the relationship among characters or groups of characters of a computer program's source instructions, independent of their meaning or manner of their use, to see that they are arranged according to the rules governing the structure of a specific language.

System Acquisition. The procurement of a weapon system by the government from an industrial contractor where typically the system consists of hardware and software integrated together as an operational system.

Target Computer. The computer in which a program is intended to execute.

Test Supervisor Software. Software resident in a target machine, which traps variable values generated during the execution of a computer program, which can be printed out or evaluated by other software to help facilitate testing (e.g., in conjunction with an APSE resident in a host computer).

Text Editor. A support computer program (tool) used for creating and revising a source program listing.

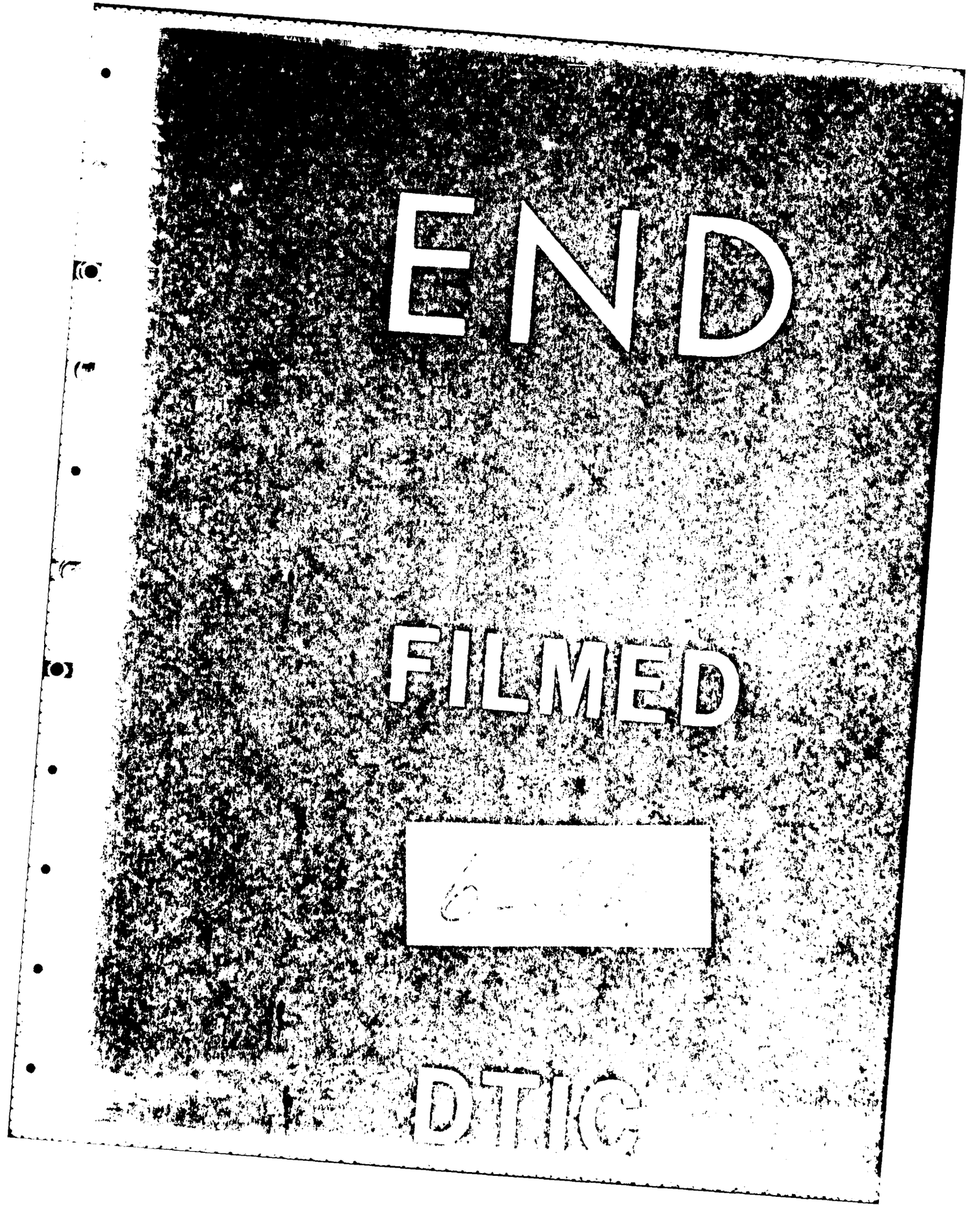
Timing Analyzer. A support computer program (tool) that estimates or measures the execution time of a computer program either by summing the execution times of the individual instructions in each path, or by inserting probes at specific points in the program and measuring the execution time between probes.

Tools. Support computer programs (e.g., an editor, loader, or debugger).

Top-Down. Pertaining to an approach that starts with the highest level component of a hierarchy and proceeds through progressively lower levels.

Utilities. Computer programs or routines designed to perform some general support function required by other application software, by the operating system, or by system utilities.

Validation. A process undertaken to ensure that an entity (e.g., a computer program language) complies with its formally stated requirements.


END
FILMED
DTIC